Comparison on local Spark distributed system, GCP and AWS for Twitter sentiment analysis

Hao Chen
Concordia University
chase19960609@gmail.com

Yilun Sun
Concordia University
yilunsun11@gmail.com

Yixuan Li
Concordia University
liyixuan4030614@gmail.com

Han Gao
Concordia University
gaohan9409@gmail.com

ABSTRACT

With the explosively growing of data for machine learning problems, distributed computing becomes necessary and productive to big data analysis [15]. Distributed system frameworks and cloud service platforms are arising due to the need for large-scale data processing. Spark, Google Cloud Platform(GCP) and Amazon Web Services(AWS) are three outstanding representatives in these areas. There are various papers on data analysis and machine learning on Spark architecture and optimization. This survey is original since we jointly analyze the pros and cons on local Spark system, GCP and AWS relatively on the same specific problem. In this paper, we implement the entire pipeline in Twitter sentiment analysis with Spark on a local machine and on GCP and AWS. Moreover, we will also share various data processing, analytic, and scalability supported by cloud services. Finally, we present a comparative view on the performance, cost and challenges for big data analysis with Spark, GCP and AWS, which could be a reference for further big data projects.

PVLDB Reference Format:
Hao Chen, Yilun Sun, Yixuan Li and Han Gao. Comparison on local Spark distributed system, GCP and AWS for Twitter sentiment analysis. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Nowadays, people shares their personal life status through few words on Twitter. Twitter has resulted in having massive data sets of information. In Figure 1, it shows all geotagged tweets in Twitter's 1% stream January 2012 to October 2018 by the most common language of tweets sent from the locations.

As Twitter was spreading rapidly across the world, we can extract the information hiding in sentences by analyzing those data sets. Such tasks need powerful processor to accomplish. As the size of the data is increasing and the complexity of the task also increases, single machine is not capable to have a good performance on processing data and tasks in a certain time. People who work in computer science come up with the idea of distributed system which helps to make data distributed and parallelize tasks.

Figure 1: 1% tweet stream 2012 to 2018 [8]

The question is that not all companies or organization have the resource to build a customer distributed system, hence later on cloud computing becomes the trend in industry. Over the past several years, cloud computing has emerged as an important new paradigm for building and using software systems [17]. This trend motivates us to conduct a comparative project in this course, and combined the distributed systems concepts with contemporary hot topic - machine learning. AI technology such as Machine Learning and Deep Learning are hot topics in many fields. Nowadays, AI and clouds represent the future, and machine learning also has a hand-in-hand relationship with large-scale data processing and distributed computing, whether in industry or academia, which would be a perfect match to our motivation.

As a proposed solution in the project, our goal is to find out the trade-off between local distributed system, GCP and AWS, and we are going to compare the difference between the performance of distributed computing such as running time, CPU utilization, Network and financial cost on Amazon Web Services and the Google Cloud Platform. These results could contribute on optimizing decision-making for to future large-scale data analytics or machine learning projects, and this could be a reference for future work on related topics.

According to [5], Sentiment Analysis is one of the most

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

simple problems in Natural Language Processing. The computing system does not need to completely perceive the semantics of each proposition but to detect the whole attitude of the author and in following classify it according to its polarity. Current existing approaches to sentiment analysis can be grouped into three main categories: knowledge-based techniques, statistical methods, and hybrid approaches. In this project, we are going to leverage Word2vec to produce word embedding and utilize continuous bag-of-words (CBOW) architecture.

In this paper, we performed a sentiment analysis on Twitter using Spark framework. We will use Decision Trees as our prediction model, and we will implement it in Apache Spark, a prominent distributed environment appropriate for processing large-scale data [5]. Then, we are going to build on our own local distributed system. A distributed computing system involves nodes (networked computers) that run processes in parallel and communicate each other. We use our PC in the same network to configure a local private cluster. After that, we install a complied spark version on local to launch the cluster script using 'cluster mode' in spark. Last but not least, we need to learn and deploy our system on Google Cloud and Amazon EC2, and record the results with respect to running time and other performance. At the end, there are evaluation metrics to measure the local cluster and cloud platforms based on performance, scalability, availability and complexity.

2. RELATED WORK

2.1 Distributed system cluster

A distributed systems cluster refers to a group of that are virtually or geographically separated and are grouped together to achieve the same service or application and can be considered as one computer.

Recently, the computational requirements for large scale data-intensive analysis of scientific data have grown significantly. In June 2009, Rafael Moreno-Vozmediano analyzed the deployment of generic clustered services on top of a virtualized infrastructure layer that combines a VM manager (the OpenNebula engine) and a cloud resource provider (Amazon EC2)[14]. In 2012, Students from University of California, Berkeley's AMPLab present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner[12]. They have implemented RDDs in a system called Spark. Same year, Lizhe Wang presents the design and implementation of G-Hadoop, a MapReduce framework that aims to enable large-scale distributed computing across multiple clusters[9]. April 2016, with the emergence of Cloud Computing, new programming models, like Spark, have appeared to suit with large-scale data processing on clouds. we investigate the applicability of Spark to develop parallel DE schemes to be executed in a distributed environment. Both the master-slave and the island-based DE schemes usually found in the literature have been implemented using Spark. The speedup and efficiency of all the implementations were evaluated on the Amazon Web Services (AWS) public cloud[4].

2.2 MLlib

Apache Spark as a highly scalable, fast and in-memory big data processing engine, has provided such independent and open-source libraries for big data machine learning which benefits from distributed architecture and automatic data parallelization [3]. In recent years, many facets of data science solutions have been amended by such a library [3]. In 2016, X. Meng, Joseph Bradley et al. [13] conducted different versions of MLlib to present the core features and the growth, and they analyzed the performance and scalability on different machine learning models on MLlib v1.1. In 2017, Mehdi Assefi, Ehsun Behravesh et al. [3] applied MLlib v2.0 on real world machine learning experiments to examine the qualitative and quantitative attributes of MLlib v2.0. Moreover, they showed a comparative view of MLlib v2.0 with Weka 3 (a GNU machine learning software) under different models experiments and data set experiments. MLlib stands out on various machine learning algorithms and data sets with respect to running time based on the comparative study demonstrated in the paper. In 2018, in the paper presented by Samar Al-Saqqa et al. [1], Apache Spark and MLlib are used for analyzing the sentiment of a large volume of online customer reviews. They purposed a similar approach pipeline as ours, and among the three experimented Spark's MLlib classifiers (Naïve Bayes Classifier (NBC), Support Vector Machines (SVM), Logistic Regression) applied on an Amazon reviews dataset, they found that SVM achieves the best performance in terms of accuracy, followed by NBC and Logistic Regression.

2.3 NLP

Sentiment analysis is an automated process of understanding a point of view from a written or spoken language to a particular topic. In the world, people generate 2.5QB bytes of data every day, and sentiment analysis has become a key tool for understanding these data.

Sentiment analysis, also known as Opinion Mining, is an area of natural language processing (NLP) that builds systems for identifying and extracting ideas in text. In general, in addition to identifying ideas, these systems extract features described, such as:

- Polarity: The speaker expresses positive or negative opinions;
- Theme: Things being talked about;
- Opinion holder: An individual or entity that expresses an opinion.

Sentiment analysis based on machine learning, [18]. First, manually mark out the emotional classification. The most positive is 4, and the most negative is 0. The feature engineering of text mainly includes data cleaning, feature construction, dimensionality reduction and feature selection. The first is data cleaning, such as to stop words, go to non-specified special expression characters, uppercase to lowercase, remove HTML tags, etc., convert special symbols, remove duplicate characters in words. Then it is the feature construction, which can construct text features based on the word bag model, such as the word frequency matrix of the vector space model, the Tf-Idf matrix, and LSA and LDA, or the text distributed features of word2vec, glove, etc. . It is also possible to construct text features with n-grams. Select the characteristic words in these texts, for example, the feature words in the emotional classification should select the emotional words, and the commodity name, the special description words of the goods, etc. should be selected

when the product classification is performed, and the bags of words model is constructed. That is, the word frequency of each word is counted to form a word matrix. Select one or several features with outstanding effects to train the model, [19].

2.4 Spark

Apache Spark has been used frequently in recent project related to sentimental analysis and machine learning. Apache Spark is an open-source distributed general-purpose cluster-computing framework[11]. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab, the Spark code-base was later donated to the Apache Software Foundation, which has maintained it since. In 2014, Carlini et al. [20] presented an adaptation to the JA-BE-JA algorithm. First, they implemented JA-BE-JA over amodified version of PeerSim. Then, they implemented a second version in Apache Spark. The experiments showed that the Spark version ran significantly faster than the former one, still providing good performances in terms ofedge-cut. In 2016, Xinan Yuan and his team use R3D [21], an RDMA-based in-memory RDD storage layer for Spark to improve the efficiency and reduce the latency in project(Figure 2). In 2018, Elias Dritsas and his team propose three classication algorithms for tweet level sentiment analysis in Spark due to its suitability for Big Data processing against its predecessors, MapReduce and Hadoop [5].

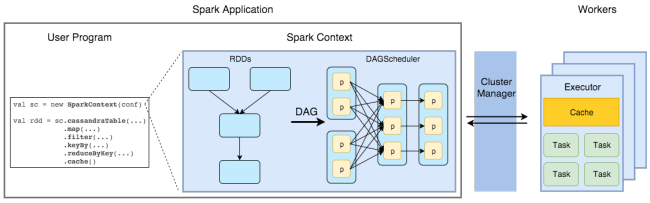

Figure 2: spark workflow

2.5 GCP

Google Cloud Platform differs from AWS, Azure and other cloud services in a various ways. It has better pricing than other cloud providers which we will compare the cost with AWS in this paper. It also integrates powerful real-time processing services, such as Cloud Dataflow, Cloud Dataproc which will also be used in our project. With GCP, users can take advantages of their security model which is an end-to-end process, but we will not cover this topic in our project. According to Google Trends website, comparing with AWS Cloud and Azure, GCP has decent interest in Asia (e.g. in China, 34% by GCP, 31% by AWS Cloud, 35% by Azure), while it has lower rate in North America as shown in the Figure 3.

2.6 AWS

Amazon's most significant advantage lies in its dominant position in the public cloud market. Part of the reason for its popularity is undoubtedly the full range of its operations. Amazon Web Service(AWS) has a broad and growing range of available services and the most comprehensive network in the global data center. AWS is a mature, enterprise-class

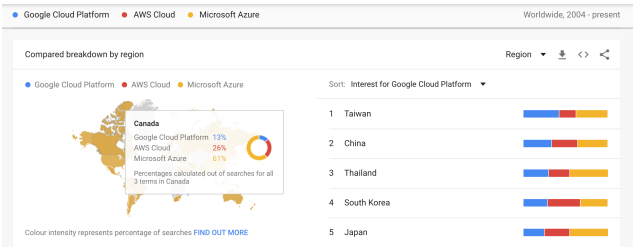

Figure 3: GCP-AWS-Azure interests in Canada

provider with the most powerful ability to manage large numbers of users and resources. Amazon's weaknesses are related to costs. While AWS regularly cuts prices, many companies find it challenging to understand the company's cost structure, and it is difficult to manage these costs efficiently when running large workloads on services. However, in general, these advantages go beyond Amazon's shortcomings, and organizations of all sizes continue to use AWS for a wide variety of workloads.

Amazon's flagship computing service is Elastic Compute Cloud or EC2. Amazon describes EC2 as "a Web service that provides secure, resizable computing capacity in the cloud." EC2 offers a variety of options, including a wide range of examples, support for Windows and Linux, bare metal instances (currently preview), GPU instances, high-performance computing, automatic scaling, and more. AWS also offers a free package for EC2, including 750 hours of t2.micro per month for up to 12 months. Among the computing categories, Amazon's various container services are becoming more popular, and it has the option to support Docker, Kubernetes and its own Fargate services, which automate server and cluster management when using containers. It also provides a virtual private cloud option called Light-sail, a batch for batch computing jobs, Elastic Beanstalk for running and extending web applications, and a few other services.

3. CURRENT STATUS

3.1 Current Progress

For now, we have done the dataset collection, pre-processing, and data exaction, the next phase is training dataset with Machine Learning by Spark.

Our dataset is from Kaggle. It contains 1,600,000 tweets extracted using the twitter API. The tweets have been annotated (0 = negative, 2 = neutral, 4 = positive) and they can be used to detect sentiment.

During the preprocessing, all irrelevant data were discarded, and we only used the actual text of the tweet, as well as the label that represents the sentiment; positive, negative or neutral. Each tweet is then tokenized and processed.

Occurrences of usernames and URLs are replaced by special tags and each tweet is finally represented as a vector which consists of the following features[2]:

– Unigrams, which are frequencies of words occurring in the tweets.

– Bigrams, which are frequencies of sequences of 2 words occurring in the tweets.

– Trigrams, which are frequencies of sequences of 3 words occurring in the tweets.

– Username, which is a binary flag that represents the existence of a user mention in the tweet.

– Hashtag, which is a binary flag that represents the existence of a hashtag in the tweet.

– URL, which is a binary flag that represents the existence of a URL in the tweet.

Then, we save the data output from NLP process into a file, then we use PySpark framework to read from this file to run machine learning task. Due to the output data has 1.6 million instances and 140 dimension, we sub-sample 20,000 instances from original data sets. The algorithm we used was decision tree. Then, we install spark on 2 nodes in the same network become a small cluster, using stand-along mode to initial one master and one worker. In local distributed system, the sub-sample running uses 20 seconds. When success in local, we deployed the system on google cloud .With the budge on trial ($300), we can only have maximum 8 cores.In order to test the performance, scalability, and reliability in google cloud,we have two different data size from 100 MB to 1GB,then we try different configuration on google cloud.The result show in Table3

Another platform use in this project is AWS,The concept about how spark works in amazon is similar to GCP ,but AWS need to manually install spark on the cluster,in this perspective,GCP is more convenient compare to AWS.The result perform by AWS has show in Table6

3.2 Contribution of each team member

In our team, the current contribution of each member is describing as the following:

Hao Chen My contribution mainly in install spark environment,configure local distributed system help team member to learn the basic information about spark. In machine learning step, creating spark dataframe in form of spark pipline and spark decision tree. In GCP, try to use different configuration to run two tasks and record the result.

Yilun Sun At the very beginning of the whole project, pre-processing of the entire project, I had applied an extensive set of pre-processing steps to minimize the size of the feature set to make it fitting for learning algorithms.

Yixuan Li In the project, I was mainly responsible for the Google Cloud deployment part. After we completed the local distributed system with Spark, I followed and reproduced the example written in Ricky Kim's article on Medium [6] which is also acted as our baseline of GCP sub-problem. After we applied for free-trial accounts and did the initial settings, we analyzed the GCP performance on different architectures and different data amount on our Sentiment data set. We also compared our GCP system with Ricky Kim's results and we will summarize the differences, similarity and results of our system and the baseline in section 6.1 and 6.2.

Han Gao In NLP data processing phase, I was responsible for feature extracting by using n-grams and wordVec approaches to maximize the performance. Then Yilun and me working together on the AWS platform which is to run the same distributed computing program with different cluster setting and record the performance and cost in the end. AWS provides a big data platform call EMR, using open source tools such as Apache Spark coupled with the dynamic scalability of Amazon EC2 and scalable storage of Amazon S3.

4. METHODOLOGY

4.1 Overall concept and idea

With large scale data, a single machine will take too long to get the result, so we need to perform task paralleled or data paralleled into different workers. In this application we are to learn and use new technology which is spark and cloud services like Google Cloud and Amazon EC2 to deal with sentimental analysis in large scale data. Then we are going to compare between distributed systems and how well they perform on the task by measuring our evaluation metrics.

4.2 System Architecture

Figure 4 highlights our project pipeline and our evaluation metrics which will be used in the project.

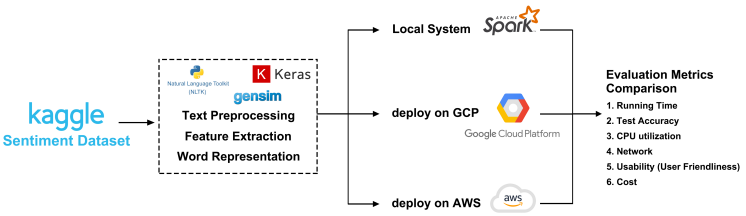

Figure 4: pipeline

4.3 Sub-problems

In the comparative project, our problems are basically divided into 5 parts:

1. Data preprocessing at the beginning
2. Implement local distributed system with Spark
3. Deploy on Google Cloud Platform
4. Deploy on AWS EC2
5. Present a comparative view over the results and performance on local, GCP and AWS EC2

4.4 Major steps of our solution

In part 1, we leverage libraries and apply a few natural language processing techniques on the original Kaggle dataset in order to get well-formatted data for applying distributed system techniques and for training. We will save the well-formatted vectors into a file which will be the expected input to the next step. We did some research on how to perform text pre-processing and feature extraction, and we discuss more about the techniques we use in this part in Section 5.1 NLP in details.

After the pre-processing part, we comes to part 2. In this part, we are going to use distributed cluster computing framework Spark on one-master-one-worker architecture on a single machine and implemented a local system on small and medium sentiment analysis data. In our experiment on local, we tried $\frac{1}{30}$ and $\frac{1}{3}$ of the whole data set, which are 100 MB and 1 GB respectively, and we finished in a reasonable time even with one third of the whole data. This part allows us to have a general sense of the complexity, performance and usability on building and using local system. Later on this will be our first benchmark on the comparison.

During part 3, we draw our next milestone on Google Cloud. First, we set up 2 free trial accounts on Google Cloud and learned a few APIs enabled on Google Cloud.

Next, we installed Google Cloud SDK on one of our machines, however this is optional since user can operate either in command line or on graphics interface. Then, we plan to do the configuration settings on cloud in order to set up the architecture, and we discuss this step by step in section 5.3. In last step, we adjust the PySpark script in order to fit the running on Google Cloud successfully. And most importantly, we vary our data size and master-worker architecture on Google Cloud and record the cloud performance for later work.

In Part 4, we are to learning and dealing with Amazon Cloud. The reason we decide to deploy our system on AWS after discussion is that:

- Firstly, it is unfair to compare a local system with cloud after we did such comparison. Since for the local system, we have installed and set up all the environment and data is downloaded on a same machine, whereas the cloud is like a black box, and we know few thing about its back-end functioning, for example, while we are doing with Google Cloud, it performs much worse on small-size data in terms of time compared with our local system. And based on our assumption, the cloud may have to spend much time on preparing the environment at the beginning for each submitted job even if the data size is small. So overall, it seems perform badly on small-size data.

- Secondly, we found that Google Cloud is not good at outputting performance chart for each job. So we will solve these questions by adding a new section with AWS Cloud, and focus on comparing the Google Cloud and AWS Cloud on their performance and ease of use of their service.

In the last part 5, we draw the attention on the comparison on GCP and AWS and collect all the results together in several charts. The purpose of this part is to present comparative results in terms of our evaluation and come up with conclusion and future work.

4.5 Assumptions

Based on the Speedup Efficiency ($SE = \frac{serial\ time}{cores * parallel\ time}$) In some condition, the more worker we have, the more efficient without changing the accuracy.

The accuracy in local distributed system and google cloud should be consistent.I hope that we can get more than 70% of the accuracy.

The system should deal with the increasing data in some extension in order to have the scalability.

4.6 Parameters in our project

In our project, we have a few parameters that fill in our evaluation metrics:

- Master-Worker architecture

  Master-worker architecture should have a large impact on the performance of distributed system. Different number of masters and workers should perform differently. And on Google Cloud we have tried different combinations and gathered the results in Table 3 in section 5.3.

- Data size

  Data size is another big factor on performance. And so far, our local system performs better both on small data ($\frac{1}{30}$ of the whole data set, 100 MB) and medium data size ($\frac{1}{3}$ of the whole data set, 1 GB).

- Cloud platform

  Due to the unfair question we mentioned in section 4.4, we will deploy our system on another cloud, Amazon Web Service, and we will mainly compare their output performance on varying data amount. In addition, we will do an analysis on the cost on both cloud platforms.

4.7 Evaluation measures

We are going to measure between local distributed system implement using Spark and GCP as same using spark, also between AWS and GCP as well. Our measurement are:

1. Effectiveness and efficiency between local distributed system and GCP and AWS.

   Effectiveness : Task performed in tree platform has the same accuracy which is 76%.Show in Figure 8. Efficiency : Efficiency in this context correspond to task execute time.The result for GCP show in table3,the result for local distributed system show in table**??**,the result for AWS show in table5.Overall,the local distributed system has the best performance,and AWS has better performance than GCP.

2. Cost: In local, the cost reflects on time effort and human resources. On cloud platforms, GCP and AWS, the cost reflects on the price cost and usability.The cost for GCP and AWS show in table4

3. Scalability : scalability refers to the ability of the system to expand the number of nodes and growing amount of data continuously. We will measure this by increasing data from around 100 MB to around 1 GB.The result of scalability related to the performance.

4. Complexity : the trade-off between the run time and the number of machines. We measure this outcome by varying the architecture on cloud platforms.

4.8 Creativity

In the project, we will create a particular cluster with 3 masters and 2 workers on GCP. As this is a new feature only supported by Google Cloud Dataproc known as Hadoop High Availability Mode (Hadoop HA). The differences between the Hadoop HA mode and default mode are:

1. Hadoop HA mode avoid the rare case of unexpected Google Compute Engine failure. When the master fails in the default single-master configuration, the in-flight jobs will necessarily fail and need to be retried.

2. As GCP documentation illustrates, in Hadoop HA mode, HDFS High Availability and YARN High Availability are configured to allow uninterrupted YARN and HDFS operations despite any single-node failures/reboots.

During the GCP experiments, we will examine with this configuration and record the results in section 5.3.

5. EXPERIMENTAL EVALUATION

5.1 NLP

Data pre-processing is the first step in the pipeline of a Natural Language Processing (NLP) system, with potential impact in its final performance. we performed simple text pre-processing like tokenizing, lemmatizing, lowercasing, then we discarded all irrelevant data and we only used the actual text of the tweet, as well as the label that represents the sentiment; positive, negative or neutral. Each tweet is then tokenized and processed.

The second step is data extraction which using the Word2vec to create feature vector where words are mapped to vectors of real numbers. There are two main training algorithms for word2vec, one is the continuous bag of words(CBOW), another is called skip-gram. The major difference between these two methods is that CBOW is using context to predict a target word while skip-gram is using a word to predict a target context. We choose the CBOW way. We implement the word2vec by Gensim. Gensim is an open source python library for natural language processing.

5.2 Local distributed system experiment

After the last step is done,we save the output into a file, then,we save the data output from NLP process into a file, then we use PySpark framework read from this file to run machine learning task. Due to the output data has 1.6 million instances and 140 dimensions, we sub sample 20,000 instances from original data sets. The algorithm used was decision tree. Then,we install spark on 2 nodes in the same network become a small cluster, due to the network reason here we only using one computer play both roles(master/worker) using stand-along mode to initial one master and one worker. In local distributed system, the sub sample running using 20 seconds.The,we scale the data to 1GB,the performance increase to 1min 2 sec,the result show in (Table 1).

Table 1: Local system Performance

Architecture	Data amount	Run time	Accuracy
1 master 1 worker	100 MB 1 GB	25 sec 1.2 min	77% 76%

5.3 GCP experiments

After we have succeed in local, we deployed the system on Google Cloud. With the budge on trial ($300), we can only use maximum 8 cores. The set up was not easy in order to run cluster service on the cloud.

Firstly, we need to create a storage bucket in order to store the data and scripts on cloud. For the sake of locality, we create our bucket in *northamerica-northeast1 (Montreal)*. Secondly, we upload the data and script to the bucket. Thirdly, we created a cluster in dataproc to set up certain settings (number of CPUs, memory, disk size) for master and workers, and general settings for the cluster (e.g. Name, Region, Zone, Cluster mode, etc.). The Region and the Zone of the cluster should be chosen to be the closest to the storage bucket in order to reduce the network delay.

Once the cluster is created, we are able to submit our job and run script on the cluster. Google Cloud Platform already installed a PySpark version and built-in machine learning package (MLlib), so we don't need to install the software. In order to run script on cloud, we need to specify relative path of our script and the data, and pass the paths as arguments. We experiment on different master-worker architecture on different amount of data, and we will present our master-worker settings (Table 2) and the results on Google Cloud (Table 3) below.

Table 2: Master and Workers configuration

Architecture	Cores	Memory	Disk size
1 master 2 workers	4 cores 2 cores/each	15 GB 15 GB	500 GB 500 GB
1 master 6 workers	2 cores 1 core/each	15 GB 15 GB	500 GB 500 GB
3 master 2 workers	2 cores/each 1 core/each	15 GB 15 GB	500 GB 500 GB
1 masters	8 cores	15 GB	500 GB

Table 3: Google Cloud Performance

Architecture	Data amount	Running time	CPU (%)	Network bytes (bytes/sec)
1 master 2 workers	100 MB 1 GB	2min 3min 43 sec	54% 48%	in1000/out200 in582/out123
1 masters 6 workers	100 MB 1 GB	1min 57 sec 3min 19 sec	8% 22%	in112/out80 in818/out133
3 master 2 workers	100 MB 1 GB	2min 11 sec 4min 20 sec	18% 22%	in78/out17 in89/out22
1 master	100 MB 1 GB	1min 28 sec 3min 12 sec	64% 68%	in1000/out189 in1214/out312

So far, we have tried data size on 100 MB ($\frac{1}{30}$ of the whole data amount) and 1 GB ($\frac{1}{3}$ of the whole data amount).

The first architecture we used on cloud is 1 master (4 cores) and 2 workers (2 cores/each). The running time is 2 minutes under this architecture on 100 MB data amount. In this case, the result is much worse than our local distributed system which is 25 seconds on the same data amount.

The second architecture we tried is 1 master (2 cores) and 6 workers (1 core each). The running time is 3 minutes 19 seconds on data size of 1 GB, and 1 minute 57 seconds on data size of 100 MB.

The third architecture we tried is 3 master (2 cores each) and 2 workers (1 core each). Google Cloud provides us this kind of architecture which marks as high availability (3 masters). And it finished in 2 minutes 11 seconds on data size of 100 MB.

In the configuration under 1 master and 2 workers running, Figure 5 shows the CPU utilization during task with 100 MB data. We can observe there is a increasing trend in CPU utilization between 10:02 and 10:03, this is due to our master is divide work to workers and doing some process. Around 10:03, the master CPU reach it's peak ,then two workers also increase the CPU utilization reach to the peak after the CPU utilization in master decreased.

Compared to our local system, accuracy achieved on Google Cloud are same as our local system. But the downsides of Google Cloud we found are:

1. Submitted job cannot be re-run again, the only way to re-run a job is to re-submit a new job.

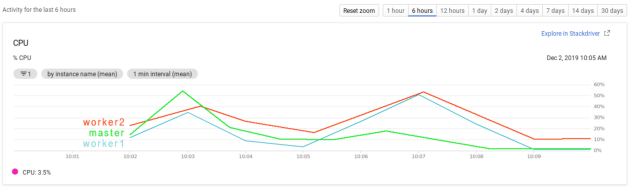

Figure 5: GCP CPU Utilization

2. In small-size data, it performs worse than local distributed system. Whereas in large-size of data, it runs in shorter time.

   Our assumptions are, on cloud, it needs to initialize environment settings for the submitted job and fetch data in another physical machine. So it takes more time at the very beginning. But once master has finished the initialization, it runs faster in the data processing and training part.

3. High cost

   With a trial budget of 300$, it is not enough for us to complete this project, so 2 of us registered 2 trial accounts in total in order to use the budget in turn.

5.4 AWS EC2 experiments

First step is to setup the setting up an EMR instance, Amazon EMR provides a managed Hadoop framework that makes it easy, fast, and cost-effective to process vast amounts of data across dynamically scalable Amazon EC2 instances. You can also run other popular distributed frameworks such as Apache Spark, HBase, Presto, and Flink in Amazon EMR, and interact with data in other AWS data stores such as Amazon S3 and Amazon DynamoDB. EMR allows us to automatically distribute jobs we want to run once we run them on our master instance. Otherwise we should be configuring and installing Spark on all nodes. Next, we need to create z EC2 key pairs so we can connect to EMR master node through SSH. Then create the EMR cluster with different number of master and worker. For the testing data and Python file, we uploaded to the S3 storage bucket. In the end, we connect to the master, download and run the python file to execute the program. This is how EMR distributed the jobs through Spark framework.

Figure 6 shows a workflow of how Amazone EMR distributes tasks through Spark. We have a short comparison of distributed Spark workloads in AWS and GCP—both in terms of setup time and operating cost. When it comes to cost, Google's service is more affordable in several ways. First, the raw cost of purchasing computing power is cheaper. Running a Google Compute Engine machine with 4 vCPUs and 15 GB of RAM will pay 0.20 every hour. An identically AWS instance will cost 0.336 per hour running EMR. Table 4 shows a brief summary.

Table 4: Price summary

Engine settings	GCP price	AWS price
4 vCPUs 15 GB RAM	0.20$/hour	0.336$/hour

The second factor to consider is the granularity of the billing. AWS charges by the hour, so we pay the full rate

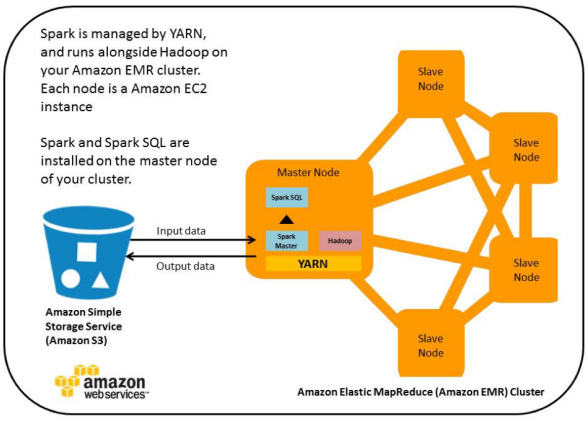

Figure 6: How EMR distributed the jobs through Spark framework.

even if our job takes 1 minutes. GCP charges by the minute, with a 10-minute minimum charge. This ends up being a huge difference in cost in a lot of use cases. From the performance table we can see this is very similar to the GCP platform, they behaved almost identically when it came to job execution time, when we are using the same setting for master and workers. The experiments on different master-worker architecture with different amount of data and the results on Amazon Web Service shown (Table 5 and Figure 7) below.

Table 5: Amazon Web Service Performance

Architecture	Data size	Run time	CPU	Network (MBs/sec)
1 master 2 workers	100 MB 1 GB	1m 23s 2m 0s	34% 45%	in0.83/out0.31 in6.06/out0.72
1 master 6 workers	100 MB 1 GB	1m 17s 2m 30s	10% 23%	in1.4/out0.86 in7.4/out1.22
1 master	100 MB 1 GB	1m 14s 3m 2s	69% 78%	in0.66/out0.41 in5.82/out2.21

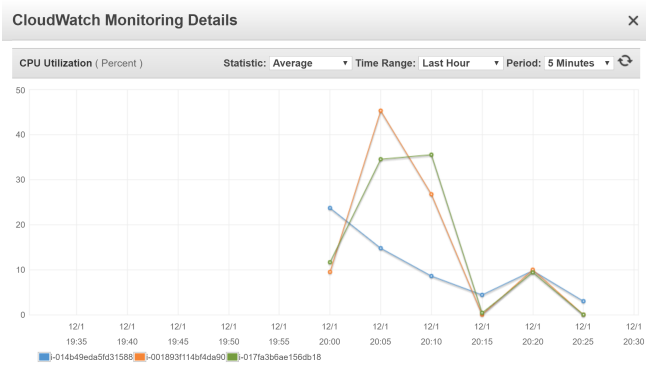

Figure 7: AWS CPU Utilization

6. BASELINE

6.1 Baseline of Google Cloud Platform

As a baseline of our project on Sentiment Analysis in Google Cloud part, in Ricky Kim's article on Medium [6], he performed a standard pipeline on Google Cloud in a data set of 1.6 million labelled tweets for sentiment analysis. In the methodology, he used an n-gram model to represent each word and a logistic regression model in PySpark MLlib for sentiment classification task. Eventually, he demonstrated a surprising result of 80.936% accuracy in 15 minutes 35 seconds on the 1.6 million data set on one master (4 cores) and two workers (2 cores/each) architecture on Google Cloud.

The reason that we choose Ricky's work as our baseline on Google Cloud is due to the following 2 reasons:

a. similar objective

Our project has a similar objective as Ricky, however, we dug more on the comparison of performance on local machines and on cloud, and difference architectures on cloud. Currently, we achieved better results in terms of time on local machines than on cloud on small-size data.

b. same amount of data

Both our dataset and the dataset in Ricky's article comes from Sentiment140 on Kaggle. During our experiment on GCP, we used the same one-master-two-worker architecture with the same cores configuration. And we achieved around 23% error rate (i.e. 77% of accuracy) in 25 seconds on a smaller data size of 98 MB, and around 24% error rate (i.e. 76% accuracy) in 1.2 minutes on a medium size of 900 MB.

In our project, we also used a different tweet pre-processing strategy and word representation, and we will present the difference and results in the next subsection. Latter in this paper, we will compare the results on Google Cloud with AWS cloud with respect to cost and performance in our evaluation metrics.

6.2 GCP baseline vs. Our GCP results

In this subsection, we will compare our GCP results and our GCP baseline in terms of using techniques, architectures and data size and performance in Table 6.

For the purpose of comparison, we only compare the results under the same architecture used in the baseline and our experiment.

As seen in the Table 6, the baseline model achieves better results whereas it uses much more time than ours. And our system compromises on the final accuracy to handle more data in shorter time.

Table 6: GCP Results Comparison

Parameter	Our results	GCP Baseline
Word Representation	Word2Vec	N-grams
Training model	Decision Tree	Logistic Regression
Architecture	1 master (4c) 2 workers (2c/each)	master (4c) 2 workers (2c/each)
Data size on training	1 GB	238.8 MB*
Accuracy	76%	80.94%
Running Time	3 min 43 sec	15 min 35 sec

(* indicates the data size is the raw data size, that is before transformed to vector representation.)

Compared with GCP baseline results and our results, since we use the same architecture as in baseline, the three parameters are: Word Representation, Training model and Data size. As shown in Table 6, accuracy does not fluctuate much as varying three parameters. However, the interesting thing is that the baseline system takes almost 5 times in terms of time to complete on data size of 219 MB than our GCP system.

After we did a analysis, the potential reasons were:

1. Different number of features in Word Representation

   In the methodology written in baseline article [7], there are 65536 features for each sentence in the training data frame. Whereas, for our implementation, we have 140 columns to represent a single sentence. So we consider this should be the significant reason for taking long time.

2. Different complexity on models

   Another parameter is the training model where we use Decision Tree, and the baseline uses Logistic Regression. Both decision tree and logistic regression can handle categorical data, but the differences are, for logistic regression, it tries to search for a single linear decision boundary in the feature space, whereas for decision tree, it tries to partition the data in feature space into half-spaces using axis-aligned linear decision boundaries [10]. Which model is more efficient is more dependent on the underlying distribution of data in the feature space. And this is difficult to analyze the data distribution in feature space since we use different dimensionality (140) to represent a sentence as in the baseline (65535).

6.3 Baseline of AWS EMR

We use Kerem Turgutlu's project as a baseline of our project on Sentiment Analysis on AWS, he performed an End-to-end Distributed Machine Learning using AWS EMR, Apache Spark (PySpark) with MillionSongs Data to predict what decade a particular song was released in[16]. For the methodology, he used the song features to perform the classification task by building a Logistic Regression model. Eventually, he demonstrated a surprising result of 84.9% accuracy in 1 minute 20 seconds on the 1.8 GB data set on one master (instance model m5.xlarge has 4 cores) and two workers (4 cores/each) architecture on AWS. we choose Kerem's work as our baseline on AWS because we both using the AWS EMR, Apache Spark (PySpark) and both performing the classification task by the Machine Learning Algorithm

6.4 AWS baseline vs. Our AWS results

Comparing our AWS results and the AWS baseline in terms of models of training technique, architectures, size of data and performance in Table 7.

For the purpose of equity, the results under the same architecture used in the baseline and our experiment will be count into comparison.

As is depicted in the results Table 7, the baseline model achieves better results.

They has created a 1D feature vector out of timbre_segments by taking the co-variances over all segments and means of each segments. Adding up to total of 90 features for 12 segments. Vector assemblers the baseline using do distributed

Table 7: AWS Results Comparison

Parameter	Our results	AWS Baseline
Training model	Decision Tree	Logistic Regression
Architecture	1 master (4c) 2 workers (2c/each)	1 master (4c) 2 workers (2c/each)
Data size on training	1000 MB	1800 MB
Accuracy	76%	84.9%
Running Time	3 min 43 sec	1 min 20 sec

training. Running the logistic regression model, but scripts provided in the repository run both logistic regression and random forest. They evaluate the model's by using multi-class F1 score since accuracy would not be good enough. Comparing AWS baseline results with our results, since we use the same architecture as in baseline, the three parameters are: Data structures, Training model and Data size. As recorded in Table 7, accuracy does not fluctuate much as varying three parameters. Although the size of the baseline system is twice of us, its running time almost half time of us.

7. ANALYSIS AND DISCUSSION

7.1 Discussion on label prediction

In this subsection, we want to show the example to our prediction as show in Figure 8. We choose 3 samples from data which corresponds to label 0,0,4. Then we transform those sentences into vectors and use decision tree to predict these value corresponding to 0,0,0 (Figure 9). In the 3 samples, there is 1 sample corresponding to the index of target 4, and 2 samples corresponding to the index of target 0. Our classifier successfully predicts 2 class-0 samples correctly, but predicts the target-4 sample incorrectly. The reason could be due to the imbalance existing in the data.

```
19966     HNDL_ Hi! Noo the sign must have fallen off. ...
15725     I have such a sore neck from sewing literally...
17730                playing wow on a saturday night !  \n
Name: text, dtype: object
[0 4 0]
```

Figure 8: original

7.2 Discussion on GCP-AWS comparison

We will summarize all results on GCP and AWS in this subsection.

Figure 10 illustrates a comparison summary with respect of running time on GCP and AWS on different architectures:

- AWS dominates GCP on time performance
- On AWS EC2, 1-master-2-worker performs almost the best
- On GCP, 1-master performs the best

Figure 11 depicts a comparison results with respect to CPU utilization on GCP and AWS on different architectures:

```
Test Error = 0.237217
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(140,[0,1,2,3,4,5...|  0.0|
|(140,[0,1,2,3,4,5...|  0.0|
|(140,[0,1,2,3,4,5...|  4.0|
+--------------------+-----+

+----------+------------+--------------------+
|prediction|indexedLabel|            features|
+----------+------------+--------------------+
|       0.0|         0.0|(140,[0,1,2,3,4,5...|
|       0.0|         0.0|(140,[0,1,2,3,4,5...|
|       0.0|         1.0|(140,[0,1,2,3,4,5...|
+----------+------------+--------------------+
```

Figure 9: predict label

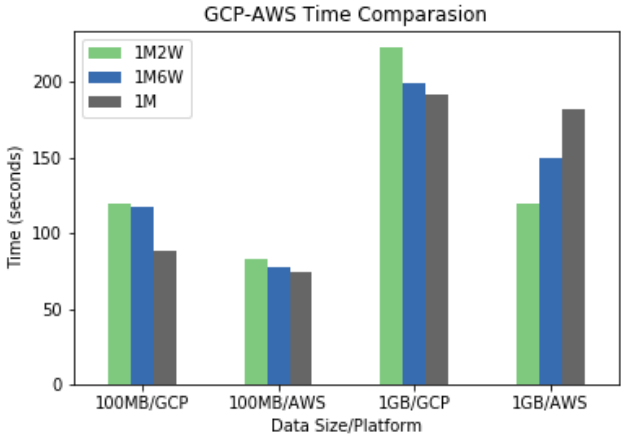

Figure 10: Run time comparison between GCP and AWS

- Under 1 master 6 workers configuration, GCP and AWS utilize CPU least, on the other hand, under 1 master configuration, both utilize CPU most
- GCP has better CPU utilization on 1 master 2 workers configuration than AWS, whereas, as data size increases, AWS catches up.

8. CONCLUSION AND FUTURE WORK

In this project, we compared the difference between the distributed computing task performing on different platforms. The main contributions of our work are to compare distributed computing in different environments and to provide a short comparison of distributed Spark workloads in AWS and GCP, both in terms of setup time and operating cost. At first, an experimental implementation of Sentiment Analysis by Spark with local computers has been performed, and second, we create the instances on GCP and the same Sentiment Analysis task has been performed on GCP. Finally, the same program execution on multiprocessors with

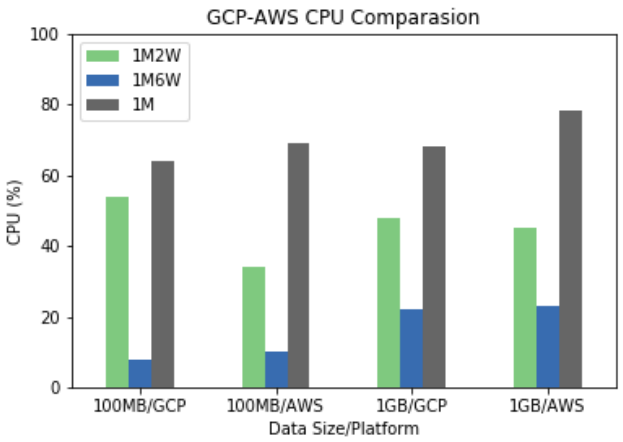

Figure 11: CPU comparison between GCP and AWS

shared memory and cluster of workstations on AWS EMR clusters.

From an experimental point of view, the distributed computing system runs on local environments can build a closed system with confidentiality, which is more suitable and safe for some tasks involving with the sensitive data. but it will cause a higher cost to purchase and maintain the servers, also the system can't provide good scalability. While the Cloud platform can assign tasks to master and workers automatically instead of assigning the role to each node in the local environment, and lower cost with great scalability.

For the AWS and GCP, in term of the financial cost, GCP it's much cheaper than AWS, First, the raw cost of purchasing computing power is cheaper. The second is the granularity of the billing. AWS charges by the hour, GCP charges by the minute, with a 10-minute minimum charge. Even the instances run for 5 mins AWS still counts as one hour. For the distributed system configurations, GCP allows multiple master nodes but AWS EMR only can set up with one master node. When it comes to the performance, although slightly different in specification, the AWS EMR has a shorter running time. Both of those two platforms provide the auto-generated monitoring detail graph.

Regarding our future work, if we have enough time and fundings for the time-consuming and costly project, enlarged scopes and dimensions of our algorithms and datasets, it may result in better performance. And considering tons of running time of neural network algorithms, which require a couple of days for the whole training process, we could deal with the optimization of reducing the computing time using spark in both could platforms. The comparison between the Google Cloud Platform and the Amazon Web Service EMR will draw more detailed conclusions.

9. REFERENCES

[1] S. Al-Saqqa, G. Al-Naymat, and ArafatAwajan. A large-scale sentiment data classification for online reviews under apache spark. *Procedia Computer Science*, 141:183–189, 2018.

[2] A. K. T. Alexandros BaltasAndreas, Kanavos. An apache spark implementation for sentiment analysis on twitter data. *LNCS, volume 10230*, 2017.

[3] M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti. *2017 IEEE International Conference on Big Data (Big Data)*, pages 3492–3498. IEEE, December 2017.

[4] P. G. J. R. B. Diego Teijeiro1, Xoán C. Pardo1 and R. Doallo. Implementing parallel differential evolution on spark. *LNCS, volume 9598*, April 2016.

[5] E. Dritsas, I. E. Livieris, K. Giotopoulos, and L. Theodorakopoulos. An apache spark implementation for graph-based hashtag sentiment classification on twitter. *Association for Computing Machinery*, November 2018.

[6] R. Kim. Pyspark sentiment analysis on google dataproc. December 2018.

[7] R. Kim. Sentiment analysis with pyspark. March 2018.

[8] K. Leetaru. Visualizing seven years of twitter's evolution: 2012-2018. March 2019.

[9] H. M. A. S. S. U. K. J. K. D. C. Lizhe Wang, Jie Tao. Mapreduce across distributed clusters for data-intensive applications. *IEEE*, 2012.

[10] V. Ma. Decision tree or logistic regression.

[11] T. D. A. D. J. M. M. M. M. J. F. S. S. I. S. Matei Zaharia, Mosharaf Chowdhury. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.

[12] T. D. A. D. J. M. M. M. M. J. F. S. S. I. S. Matei Zaharia, Mosharaf Chowdhury. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. *ACM*, April 2012.

[13] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, January 2016.

[14] R. S. M. Rafael Moreno-Vozmediano and I. M. Llorente. Elastic management of cluster-based services in the cloud. *ACM*, 2009.

[15] S. Tang, B. He, C. Yu, Y. Li, and K. Li. A survey on spark ecosystem for big data processing. November 2018.

[16] K. Turgutlu. End-to-end distributed ml using aws emr, apache spark (pyspark) with millionsongs data.

[17] P. Upadhyaya, M. Balazinska, and D. Suciu. How to price shared optimizations in the cloud. *Proceedings of the VLDB Endowment*, 5(6):2150–8097, November 2012.

[18] M. Z. S. W. G. C. T. K. N. B. C. O. J. S. M. R. R. Wei Wang, Jinyang Gao. Machine learning as an analytics service system. *Proceedings of the VLDB Endowment*, 12(2):128–140, 2018.

[19] B. G. A. H. G. M.-h. H. O. W.-C. T. Xiaolan Wang, Aaron Feng. Scalable semantic querying of text. *Proceedings of the VLDB Endowment*, 11(9):961–974, 2018.

[20] B. Yan, Z. Yang, Y. Ren, X. Tan, and E. Liu. Microblog sentiment classification using parallel svm in apache spark. *IEEE*, 2017.

[21] X. Yan, B. Wong, and S. Choy. R3s: Rdma-based rdd remote storage for spark. *ACM*, 2016.