

Big Data Processing Model for Authorship Identification

**Toh Chin Eng^{1,2}, Shafaatunnur Hasan^{1,2}, Siti Mariyam Shamsuddin^{1,2}, Nur
Eiliyah Wong² and Intan Ermahani A Jalil³**

¹UTM Big Data Centre

Universiti Teknologi Malaysia Skudai 81310 Johor

²Faculty of Computing, Universiti Teknologi Malaysia, 81310 Skudai Johor
Email: shafaatunnur@utm.my, mariyam@utm.my, nureiliyah@utm.my

³Faculty of Information and Communication Technology,
Universiti Teknikal Malaysia Melaka, 76100 Melaka
ermahani@utem.edu.my

Abstract

The era of Big Data has arrived and an average of about quintillions of data is produced daily. Data can be in many forms such as image, document or movie. For document file, there are digitalized document and handwritten document that often relates to the issue of copyright or ownership. This is due to improper authentication that leads to unhealthy authorship claimed of that particular handwritten document. Authorship identification is a sub-area of Document Image Analysis and Identification (DIAR). DIAR aim is to analyze and identify document authorship. However, for big scale of documents text images, the issue of document processing time becomes crucial for better authorship identification. Therefore, in this study, we propose an alternative solution to solve the above problems dealing with massive amount of document text images by integrating Hadoop MapReduce and Spark's MLlib for authorship identification through data processing parallelization. MapReduce processing is used as the platform to pre-process these huge document text images in Hadoop Distributed File Systems (HDFS), follows by the authorship identification through Apache Spark machine learning library. The experiments show the integration is successfully implemented for big size of document text images. However, further improvement is needed for the post-analytics of the reduced document text images for better identification.

Keywords: *Big Data, Hadoop MapReduce, Spark's MLlib, Authorship Identification, handwritten text*

1 Introduction

The era of Big Data has arrived due to advancement of hardware technologies and cheaper devices in the future. There are about 2.5 quintillion bytes of data that are created from all around the world and 90 percent of the data in the world today mostly were produced within the last four years [1]. According to International Data Corporation's (IDC) annual Digital Universe study, the amount of data on our planet is set to be reached about 44 zettabytes (4.4×10^{22} bytes) by 2020 which would be ten times larger than it was in 2013 [2]. Through observation, we can predict that the data will be kept increasing in the future and in various forms. Therefore, the traditional data processing approaches are unable to process those data with high efficiency and performance. Big data has the characteristics where it has a large volume, heterogeneous, autonomous sources with distributed and decentralized control, and seeks to explore the complex and evolving relationships among data as claimed under Heterogeneous, Autonomous, Complex and Evolving (HACE) Theorem [1]. New methods or approaches are required to be implemented in the data processing applications to solve different issues in Big Data because the traditional approaches are not suitable anymore.

Data will keep on growing at an exponential rate in the era of 4th Industrial Revolution due to gigantic trends of digital documentation in industries and government agencies in the future. Nowadays, documents can be classified into various types of format such as pdf (Portable Document Format) and .docx (Document Extended). The collection of documents and other types of data contribute to Big Data as the size of the data keeps on increasing every year and hardly to process and analyze. Since then, many types of processing engine and model have been innovated and developed to deal with big data such as Hadoop MapReduce and Apache Spark. There are many techniques that have been developed for document authorship identification using current technologies. However, when the document size becomes bigger, the efficiency to identify multiple authors' writing becomes slower. Thus an improvement must be developed for the pre – processing and post – processing of big document text images to increase the efficiency of identifying multiple authors' writing style.

Hadoop MapReduce is one of the alternative solutions in solving authorship identification. It is a new computer architecture for real – time data intensive processing which can be executed on high – performance cluster [1]. It has rapid growth and has been becoming a standard in both industry and academia [3]. It is also an open source implementation framework that is proposed by Google [3]. The main reason that has contributed to the popularity of using Hadoop MapReduce is due to its ease of use, failover, and scalability properties. The processed data will

be copied and stored in different nodes for backup to prevent any incident that might happen in the cluster such as crashing. In Hadoop ecosystem, four modules are involved: a)Hadoop Distributed File System (HDFS), b)MapReduce, c)YARN and d)Common. HDFS is a file system that can store a large amount of data with a master – slave architecture. It provides the feature of fault tolerance by making a few copies of data in the data block to prevent fault such as disk failure. While, MapReduce is a parallel programming model which is also the data processing engine which consists of map phase and reduces phase for processing data [2].

There is also another Big Data processing model which is Apache Spark. Spark is one of the Apache top level projects where it supports the iterative computation and utilizes in – memory computation to improve on speed and resource issues [2]. Spark is an open source platform for large – scale data processing that is suited for iterative machine learning tasks [4]. It's processing speed is proven when it won the Daytona GraySort Benchmark Contest by sorting 100 TB of data on 206 nodes in 23 minutes [2]. Spark has the advantage of faster processing speed although it does not provide the data storage feature as in Hadoop. Spark offers Spark Streaming (micro – batching) for real – time data processing in sequences through the batch processing [2].

Authorship identification can be defined as a process of identifying the most presumably author or writer of a disputed or anonymous document, based on a collection of known documents [5]. In this research, we focus on the big scale of handwriting text images for pre–processing and post – processing using Hadoop and Apache Spark to perform authorship identification.

The paper as follow: Section 2 gives the related works to Hadoop MapReduce, Apache Spark and authorship identification. Section 3 provides the proposed integration of Map Reduce for Authorship Identification. Section 4 explains the results, analysis and discussions based on the output and finally, Section 6 concludes our work together with future work.

2 Related Work

Hadoop MapReduce framework not just a parallel programming model but has been received significant attention due to its uniqueness and functionalities [6][7]. MapReduce framework enables the user to write functional codes for map and reduce phases and the execution will be handled by the framework. The input files will be split into many chunks where the size can be defined by the user and distributed to the worker nodes in the cluster for processing [6]. If there are any failures occur in the worker node, the task will be re – scheduled with the dynamic scheduling mechanism in MapReduce framework [8]. Automatic load balance between the worker nodes also will be provided with this approach. Finally, the data is replicated and stored in different worker nodes for the backup purpose [8]. Therefore, we can be ensured that the processed data will not be lost directly when a failure occurs and those operations are executed automatically. There is one

component in the architecture of Hadoop which is called as Hadoop Distributed File System (HDFS) [9]. The components in HDFS are Name Node, Secondary Name Node and Data Node [9]. Name Node is responsible for controlling all the Data Nodes, store and manages all the metadata of the file system which is named as *fsimage* [9].

To perform writer identification or authorship identification, proper neural architecture and machine learning schemes must be selected to deal with the representation of the data for pre – processing document text images. In Document Image Analysis and Identification (DIAR), the pre-processing tasks include character thinning, noise reduction, binarization, and skew detection [10]. The purpose of the pre – processing is to extract more information from the handwritten text and increase the accuracy of authorship identification. Artificial neural network (ANN) is one of the good machine learning algorithms that can be used for authorship identification due to its learning ability in adapting different kinds of noise [10].

There are two types of categories to perform writer identification which is text – independent method and text – dependent method [11]. Text – independent method is where any text also can be used to identify the writer while text – dependent method requires the writer to write the same text to perform writer identification [11]. The size of the input is standardized in terms of width and height for binarization using the global threshold [11]. Then, feature extraction algorithms are implemented to extract the texture features from the input data. This is because different writers have a different visual appearance for their writing [11]. However, as the number of writers is increasing, the classification rate will be lower because of higher possibility to classify wrongly [11].

There are many methods to perform authorship identification such as using stylometry to analyze author's style of writing. Four main methods are commonly used to analyze the writing: lexical, syntactic, and semantic and the content specific method. The lexical method is analyzing the document word distribution and word count in the text while the syntactic method is where the features such as frequency of words and punctuation are extracted for analysis. However, there is a limitation on those methods. The limitation is where the minimum of the data size must at least 10 times than the word count of the anonymous text file [5].

While discussing the methods to perform writer identification, some issues need to be addressed: 1) to find the mean of main features from different handwriting styles or similar handwriting styles to identify the real writer [12]; 2) to find out the significant features when comparing the handwriting [12]. By using the traditional framework, the handwriting dataset will go through feature extraction processes. In the new framework, there is a new process after feature extraction which is called

as feature discretization which extracts the similar feature into the standard representation [12]. From the analysis, it is proven that the accuracy of classification is higher when compared with the non – discretized data [12].

The final process of authorship identification is the classification stage using some artificial intelligence (AI) methods such as Artificial Neural Network (ANN). In ANN, the input image must be transformed into a sequence of feature vectors and then the image is applied with sliding window of square cell [13]. The output is passed to optical model and follows by tree lexicon process. This identification model is the combination of HMM (Hidden Markov Model) and ANN (Artificial Neural Network) [13]. The identification system is modified with a token passing algorithm to record the n – best work endings at each of the sliding window position.

Although there are many algorithms that can be used to perform authorship identification, for large – scale authorship documentation, the efficiency becomes an issue since the extracted data is not processed in parallel form. However, this issue has been partially solved since many AI algorithms and tools have been developed and improved to support the parallel processing mechanism. Spark is one of the AI tools which is Apache top level projects that supports the iterative computation and in – memory computation to improve the speed and resource issues [2]. Spark is an open source platform for large – scale data processing that is suited for iterative machine learning tasks [4]. The machine learning algorithms are grouped under Spark’s MLlib (Machine Learning Library). This module offers functionality such as collaborative filtering, optimization, feature extraction, regression, statistics, Frequent Pattern Mining, classification, Dimensionality Reduction and Clustering [2]. In general, MLlib has the goal to make the machine learning to be easy and scalable by providing common learning algorithms and utilities for performing different functionalities.

Spark has the advantage of ten times faster for the on – disk operation and hundred times faster for in – memory operation when comparing with Hadoop [4]. The integration of Hadoop and Spark can take advantage of Hadoop Distributed File System (HDFS) for storage purpose and utilize the high – speed processing power of Spark on processing [4]. The result has proven that the integration of Hadoop and Spark technologies have a higher efficiency when compared with the system that Hadoop – based only [4].

3 The Proposed Hadoop-Spark MLib Framework for Authorship Identification

The main goal of this study is to determine the availability of Hadoop Map Reducing processing on authorship identification. The study is carried out on Hadoop MapReduce framework and authorship identification. Our proposed integration of Map Reducing processing approach and Spark's MLib for authorship identification will be a new approach to performing large – scale authorship identification which is related to Big Data. Hadoop MapReduce framework will be used to handle the pre – processing part on the input data before the authorship identification is performed with Spark. OpenCV is integrated with MapReduce framework to process the input image data. Fig. 1 illustrates the process involves in developing the proposed integration



Fig. 1: Hadoop-Spark Framework for Authorship Identification

For the integration of Hadoop MapReduce framework with Spark's MLib, the built in neural network in Spark's MLib will be used to perform authorship identification. The comparison between the performance of Map Reducing processing and without Map Reducing processing on authorship identification is

carried out to investigate the efficiency of the integration. The time taken to complete authorship identification will be taken as well.

3.2 Writer Document Text Images for Hadoop Map Reduce

Hadoop MapReduce framework can accept multiple files as input with each map will assign to each input data. If there are 1000 of inputs data, then 1000 map tasks will be created to process the input data. Due to the hardware limitation for most users, not every machine can execute all map tasks concurrently. Therefore, the proper method to parse the input to MapReduce framework is vital to overcoming the problem mentioned. Sequence Files Formats is one of the methods to parse the input where all the inputs are converted to flat files that consist of binary key / value pairs and pass as input to map phase one by one.

In this study, we use total input data of 115, 320 handwriting image files. Each of the handwriting image files contains one English word. Those image files are the sample handwriting that is extracted from different documents. Those sample handwriting image files are belonging to 15 writers and all the image files are categorized well in different folders. Each writer is identified through his styles of writing. However, there exist corrupted files in the folders that must be pre-process, which lead to a new total of 115, 318 handwriting image files. The corrupted image files are detected when all the image files are converted to HijiImageBundle (HIB). HIB is a collection of images represented as a single file on HDFS [14].

In our study, Hadoop Image Processing Interface (HIPI) is implemented to parse the input data. HIPI is an image processing library that is designed with Apache Hadoop MapReduce to facilitate efficient and high-throughput image processing with MapReduce – style parallel programs executed on a cluster environment [14]. HIPI also supports Open Source Computer Vision Library (OpenCV) integration with hadoop mapping at the pre – processing process.

4 The Proposed Integration of Hadoop-Spark MLib

Big Data can be in two forms which are unstructured form and structured form. In this study, the data format for authorship documentation is in the unstructured form. Even, the data is in the structured form, the processing will be very long since the process doesn't involve parallel processing or distributed processing. Therefore, Hadoop MapReduce framework can be used for the pre – processing dealing with the massive amount of input data due to its parallelism.

In our proposed design, map and reduce functions are integrated with OpenCV to pre – process the handwriting image files such as cleansing of data and dirty data handling. The processed handwriting image files are stored in the HDFS for authorship identification. The handwriting image files are separated into 7 folders to implement k – fold validation during the authorship identification phase. One

folder is used as testing data and six folders are used as training data and this process is repeated 7 times.

The feedforward artificial neural network in the Spark's MLlib which is also called as multilayer perceptron (MLP) classifier is used for classification purpose. We define four variables to implement MLP classifier which is for input layer, two intermediate layers and output layer. The number of nodes for the output layer represents the number of classes for identification purpose [15]. The MLP classifier can be written in matrix form with $K + 1$ layers where \mathbf{w} is node's weight and \mathbf{b} is a bias term represents as follows [15]:

$$y(\mathbf{x}) = f_K(\dots f_2(\mathbf{w}_2^T f_1(\mathbf{w}_1^T \mathbf{x} + b_1) + b_2) \dots + b_K) \quad (1)$$

The nodes in the intermediate layers use the sigmoid (logistic) function:

$$f(z_i) = \frac{1}{1 + e^{-z_i}} \quad (2)$$

While the nodes in the output layer implement the softmax function:

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^N e^{z_k}} \quad (3)$$

MLP classifier uses the backpropagation approach for learning model, L – BFGS (Limited – memory Broyden – Fletcher – Goldfarb – Shanno) algorithm as the optimization routine and logistic loss function for the optimization purpose [15]. Multi – core processing is carried out throughout the whole process in a single machine with standalone cluster.

Fig. 2 illustrates the integration process starting from data collection of 115, 320 image files downloaded from IAM Handwriting Database. These datasets are stored in the same folder with many subfolders. The data consists of different handwriting styles from 15 different writers.

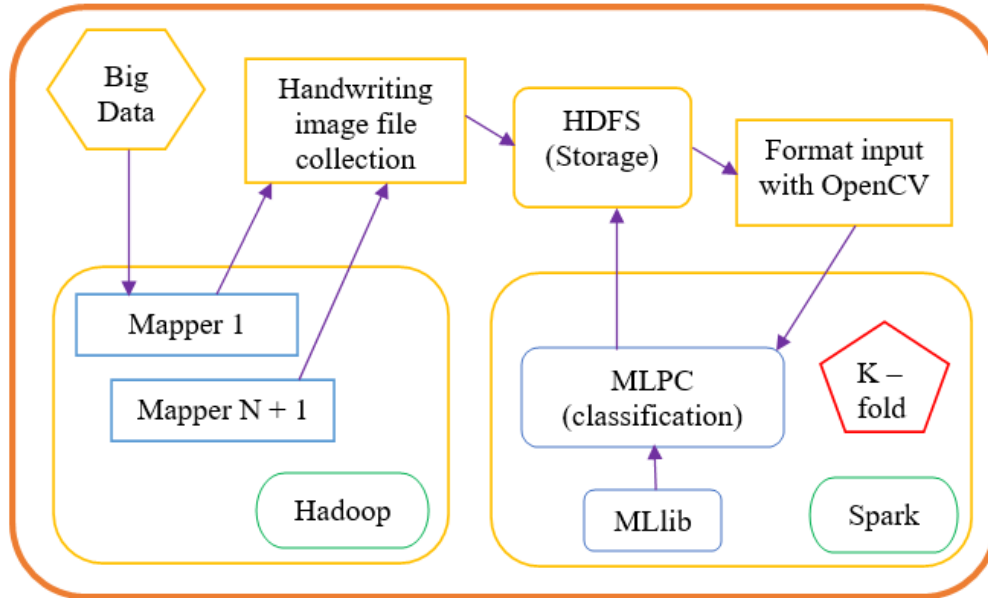


Fig. 2: The Proposed Integration of Hadoop-Spark MLlib

The input data is processed with MapReduce framework by passing through two phases: a mapping phase and reducing phase. Both phases are integrated with OpenCV for image processing process. However, due to certain issue occurred which will be explained in the next section, the map and reduce phase are integrated to perform pre – processing. The handwriting image files are de-noised for data cleansing using OpenCV. Noise is generally can consider as a random variable with zero mean in an image and must be removed to avoid inefficiency during identification process [16].

De-noise images are converted to the PGM (Portable Gray Map) image file format to obtain the lowest denominator grayscale file format and easier for reverse engineering process. Hence, we can ensure the handwriting image file is “*clean*” grayscale form. Next, the handwriting image files are binarized and rescaled to a fix width and height.

The output from the MapReduce framework is stored in the HDFS and separated into 7 folders to prepare for k – fold validation process for authorship identification. Files separation is implemented by using the python script to distribute the handwriting image files. Each folder has approximately 14,225 to 14,229 of handwriting image files. The pixel data in each of the image is extracted and saved as a single text file. The numbers will be kept as the input for the identification process.

The feedforward artificial neural network with backpropagation learning model in the Spark’s MLlib is used to perform authorship identification which is also called as Multilayer Perceptron Classifier (MLPC). The formatted handwritten image file

collection will be the input of this classifier for authorship identification. The training data set is fed to the MLPC to identify the writer's handwriting. There are 6 directories in the training set and 1 directory in the testing set. The process is executed for 7 times where each of the directories in the training set is put into the testing set to perform k – fold validation in authorship identification phase. The overall workflow for this phase is described in Fig. 3.

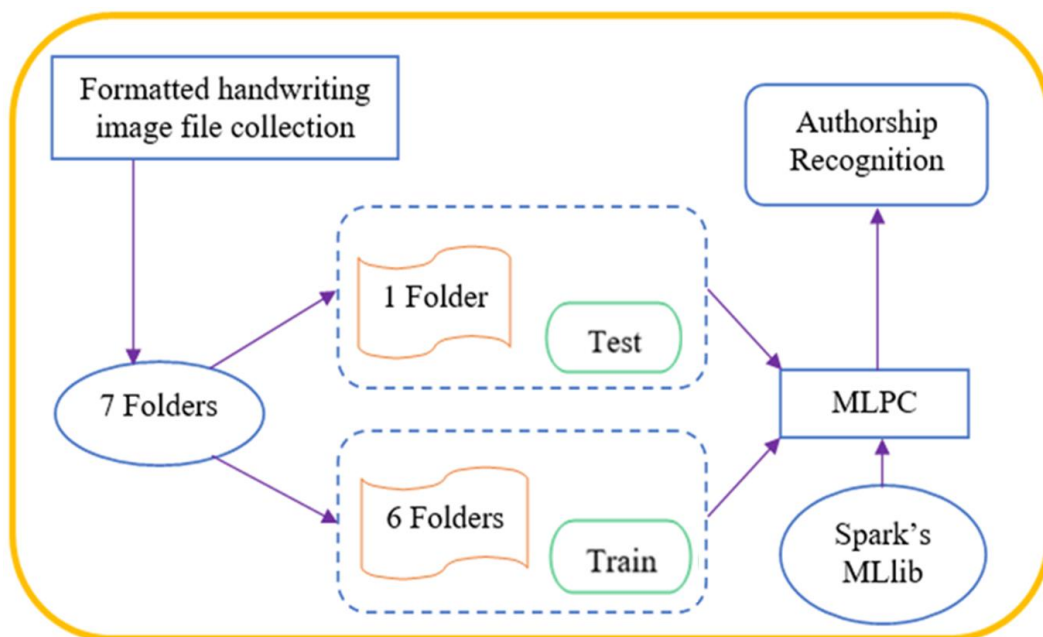


Fig 3: Authorship Identification Process in Hadoop-Spark MLlib Framework

The main focus of this research is on the integration of MapReduce framework and Spark's MLlib to perform authorship identification. However, the whole process of the proposed framework as in Figure 1 must be executed accordingly since it involves two main stages which are MapReduce and Spark processing. The first experiment is tested on the performance with Map Reduce on large scale authorship identification. While the second experiment is performed without MapReduce processing in which the mapping and reducing processing is integrated into map process only. The performance of the two experiments will be computed and the results are recorded for analytic purpose.

5 Experimental Results and Analysis

Fig. 4 shows the original image (left) with noise and image on the right side is the image after denoising. There are 4 techniques that can be used for denoising using Non-Local Means Denoising algorithm which produce better results despite their slower processing [16].



Fig. 4: Left (original) and Right (pre – process)

The image is converted to the PGM image file format after denoising process, and follows by the binarization procedure. If the pixel value in the handwriting image file exceeds the specified threshold value, the pixel will be assigned to white or black. Figure 5 shows denoised image (left) and the binarization image (right).

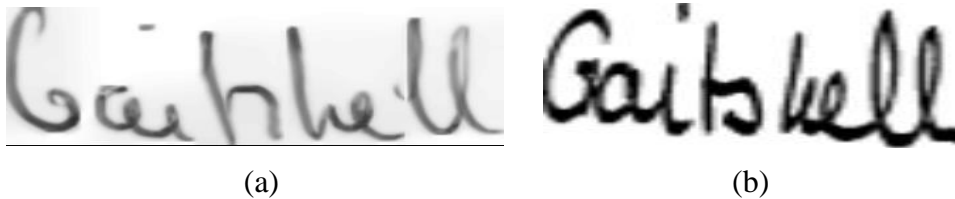


Fig. 5: Image after denoised and binarization process

Adaptive thresholding is implemented in this study since the image might have different lighting condition in different areas. The algorithm is adaptive in computing the threshold for a small region and different threshold for a different region of the image [16]. In the documents, all the handwriting image files are different in size. Thus, these files are required to be rescaled to a fix width and height for standardization and to preserve the quality of the handwriting image file.

5.1 Map Reduce for Handwriting Image Files

The handwriting images are tested ranging from small size to larger datasets. The main problem arises during the experiments is due to the broken pipe exception in which the MapReduce streaming is terminated prematurely after the mapping phase. In this scenario, the small dataset is successfully streaming without error but it doesn't work for large dataset. After multiple testing times, the threshold for processing image files is less than 16,381 documents (See Fig. 6).

```

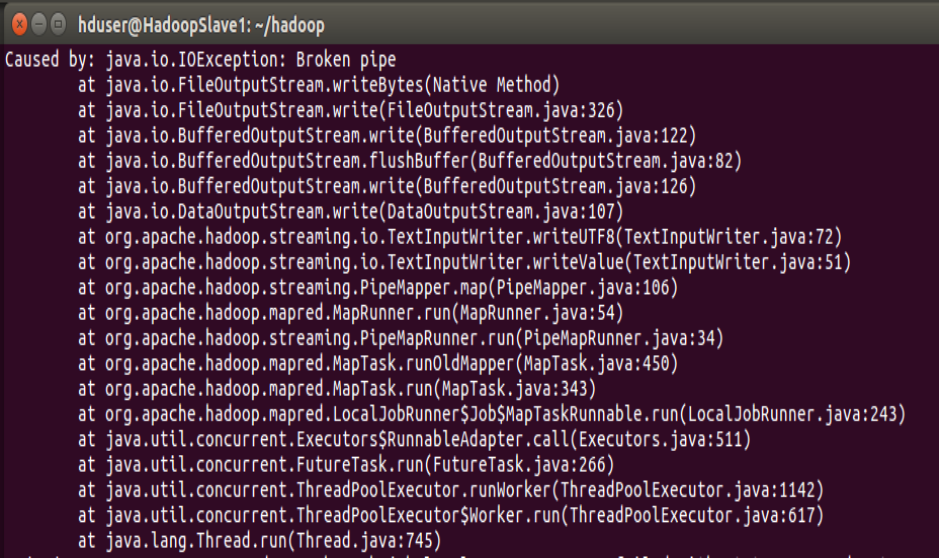
hduser@HadoopSlave1: ~/hadoop
00_0' to hdfs://localhost:54310/user/hduser/hadoop_tmp/_temporary/0/task_local200517929_0001_r_000000
17/05/07 15:28:24 INFO mapred.LocalJobRunner: reduce > reduce
17/05/07 15:28:24 INFO mapred.Task: Task 'attempt_local200517929_0001_r_000000_0' done.
17/05/07 15:28:24 INFO mapred.LocalJobRunner: Finishing task: attempt_local200517929_0001_r_000000_0
17/05/07 15:28:24 INFO mapred.LocalJobRunner: reduce task executor complete.
17/05/07 15:28:25 INFO mapreduce.Job: Job job_local200517929_0001 completed successfully
17/05/07 15:28:25 INFO mapreduce.Job: Counters: 38
File System Counters
  FILE: Number of bytes read=210308
  FILE: Number of bytes written=737522
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=262144
  HDFS: Number of bytes written=0
  HDFS: Number of read operations=13
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=4
Map-Reduce Framework
  Map input records=2814
  Map output records=0
  Map output bytes=0
  Map output materialized bytes=6
  Input split bytes=98
  Combine input records=0
  Combine output records=0
  Reduce input groups=0
  Reduce shuffle bytes=6
  Reduce input records=0
  Reduce output records=0
  Spilled Records=0
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=13
  CPU time spent (ms)=0
  Physical memory (bytes) snapshot=0
  Virtual memory (bytes) snapshot=0
  Total committed heap usage (bytes)=1373110272
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=131072
File Output Format Counters
  Bytes Written=0
17/05/07 15:28:25 INFO streaming.StreamJob: Output directory: hadoop_tmp

```

Fig. 6: 16,381 of handwriting image file

We provide the screenshot of the broken pipe exception error after the mapping process as in Fig. 7 when the number of image files is more than 16,381. In this

context, the reduce phase cannot be carried out and the streaming process is stop. Hence, all the handwriting image files cannot be pre – processed accordingly.



```

hduser@HadoopSlave1: ~/hadoop
Caused by: java.io.IOException: Broken pipe
  at java.io.FileOutputStream.writeBytes(Native Method)
  at java.io.FileOutputStream.write(FileOutputStream.java:326)
  at java.io.BufferedOutputStream.write(BufferedOutputStream.java:122)
  at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:82)
  at java.io.BufferedOutputStream.write(BufferedOutputStream.java:126)
  at java.io.DataOutputStream.write(DataOutputStream.java:107)
  at org.apache.hadoop.streaming.io.TextInputWriter.writeUTF8(TextInputWriter.java:72)
  at org.apache.hadoop.streaming.io.TextInputWriter.writeValue(TextInputWriter.java:51)
  at org.apache.hadoop.streaming.PipeMapper.map(PipeMapper.java:106)
  at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:54)
  at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
  at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:450)
  at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
  at org.apache.hadoop.mapred.LocalJobRunner$Job$MapTaskRunnable.run(LocalJobRunner.java:243)
  at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
  at java.util.concurrent.FutureTask.run(FutureTask.java:266)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
  at java.lang.Thread.run(Thread.java:745)

```

Fig. 7: Broken pipe exception issue

The causes for the failure are due to: 1) MapReduce framework initially is designed to process key value pair of data instead of image processing; 2) MapReduce framework is trying to feed the output from mapper through STDIN (command – line arguments) to reducer script. However, in this research, our mapper does not return any output in command–line arguments to the reducer phase. Therefore, the process is unsuccessfully because the input from mapper is null. Finally, the timeout for the reducer to wait for mapper output which is by default is 600 seconds in MapReduce framework. This possibility is eliminated after repeated testing by overwrite the timeout variables and delay variables for map and reduce phases, however the problem still exists. Due to these problems, we propose an improved in dealing the process of mapping writer documents for authorship identification in the next section.

5.2 The Proposed Integration of Mapper and Reducer Script for Writer Documents

Due to the above issue, we propose an integration of mapper and reducer into mapper script only. This is done as follows: 1) the image processing functions in the reducer script is moved to mapper script; 2) rewrites the mapper script by implementing the multiprocessing module and set the number of reducer task to 0

because without the reducer, the map phase output will be directly stored in the specified directory. This is where the sorting, shuffling and aggregation of output from mapping phase will not be performed by MapReduce framework. By default, those processes will be executed automatically by MapReduce framework during reducing phase. In this research, those operations are not needed for image processing. Therefore, the time taken to complete the whole process will be shortened and the streaming process can be successfully completed without error.

In this experiment, we set up MapReduce framework environment in the virtual machine. Quad cores are assigned to the virtual machine and the utilization of each core are shown in Fig. 8 and Fig. 9, respectively. The utilization of each core is more than 80% during the streaming process. The number of the final output is stored in the directory with total handwriting image files in PGM format of 99,589. These files will be used for the next phase and the other 15, 731 handwriting image files are removed. These files are corrupted image files and some of the image files are too small that cannot be used for the authorship identification purpose. Both methods are tested with 115, 318 handwriting image files as input to the proposed integration.

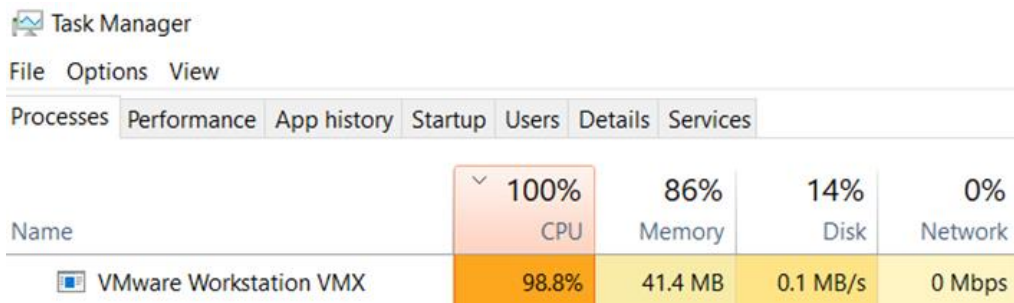


Fig. 8: CPU utilization by virtual machine

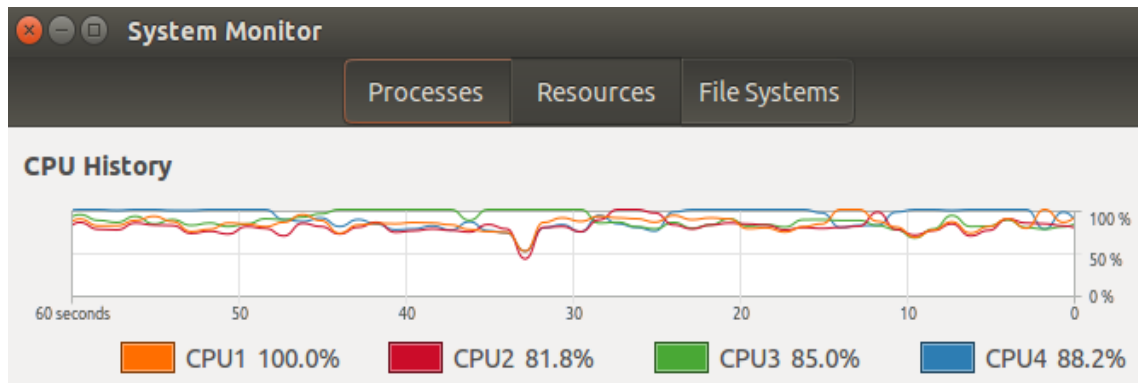


Fig. 9: CPU utilization of each core by virtual machine

The time taken to complete the processing is approximately 3 hours 15 minutes to 3 hours 20 minutes depending on the machine's processing speed. This analysis is obtained after repeatedly testing with full dataset in tandem with changing of the scripting. Various measurements are done to capture the performance of the proposed integration and these include the process of increasing the number of map tasks, implementing the multiprocessing module and increasing the number of memory allocation for the MapReduce framework during processing. These actions can be achieved by overwriting the default configuration of the framework.

Some of the variables in the MapReduce framework default configuration are changing accordingly. For example, the variable "*mapred.child.java.opts*" is set originally to 200MB and has been changed to 1024MB. This variable is referred to the java heap memory for processing MapReduce framework. The reason for changing this variable is due to frequent error message of Java heap space out of memory when dealing with large dataset; thus by changing the variable setting helps resolve the above issue.

5.3 Analysis of the Proposed Integrated Mapper and Reducer Script for Writer Documents

The handwriting image files with 76 directories are separated into 7 directories to perform k -fold validation during authorship identification phase after the proposed integration. For each of the testing process, the identification rate and duration are displayed as shown in Fig. 10. The image processing method that is used for identification purpose is based on pixel based, where all the pixel data of an image is extracted and converted to binary form. However, Spark does not offer the image processing library that can be integrated with Spark's MLlib to perform authorship identification. Due to different word length of each image, pixel data based

approach is not convincing in recognizing the word that shares the same uniqueness during learning process for identification purpose. More layers of image processing methods need to be implemented to increase the identification rate. The user can adjust the configuration of the multilayer perceptron classifier to increase the identification rate of the input data based on the specification of the machine that is used to perform the authorship identification.

```
Recognition rate: 0.18983694124262018  
Duration: 10176.09899553
```

Fig. 10: Identification rate and execution time

Fig. 11 shows the Spark's job execution from the web UI. In our study, the scheduling mode is changed to FAIR to ensure equal jobs distribution to all Spark's workers. The number of tasks and the duration to complete all the tasks on each stage is displayed to the user for tracking purposes. From the event timeline, the user can observe the running task with the status: succeeded, failed or running.

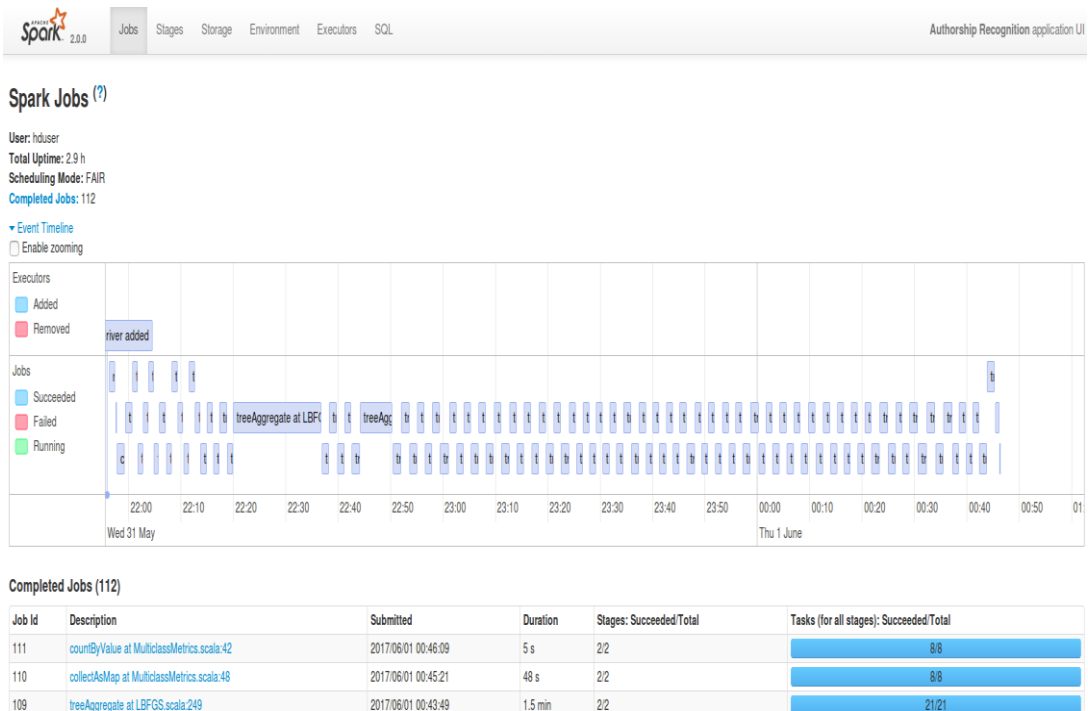


Fig. 11: Spark’s jobs

The details of the Spark’s executors or can be called as Spark’s workers can be viewed in Fig. 12. Each core is assigned to each Spark’s worker, thus 4 Spark’s workers are implemented all the jobs accordingly. The number of completed task, the total number of task and size of the input also is shown. The CPU, memory and disk utilization by Spark’s jobs are illustrated in Fig. 13.

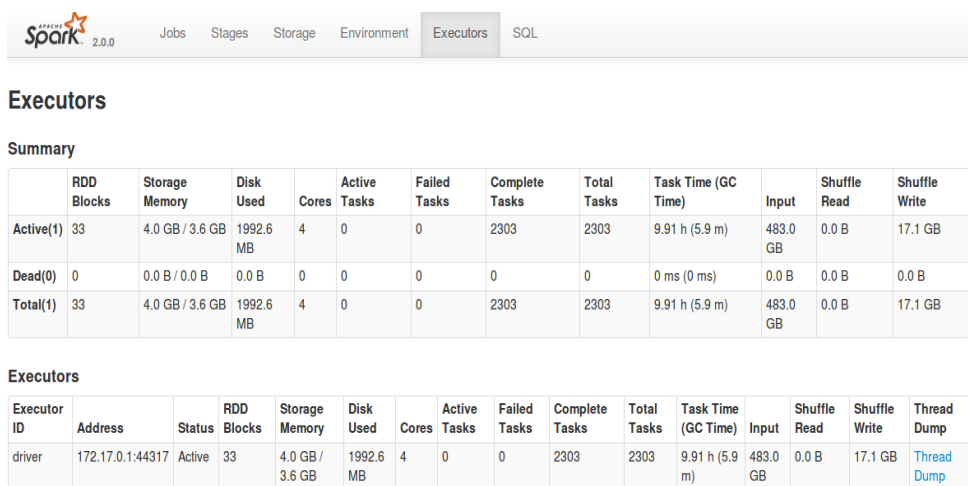


Fig 12: Spark’s executors

Name	100% CPU	91% Memory	100% Disk	0% Network
> Service Host: Superfetch	0%	98.4 MB	0.1 MB/s	0 Mbps
VMware Workstation VMX	99.5%	67.9 MB	12.2 MB/s	0 Mbps

Fig. 13: Spark’s CPU, Memory, and Disk utilization

5.4 Comparison between Map Reducing Processing and the Proposed Integration of Mapper and Reducer Script for Writer Documents

MapReduce framework is used in processing parallel the image files. First, the image processing processes are separated into the map and reduce phase. MapReduce framework can perform exceptionally well on data mining with *<key, value>* pair data but not for image processing. This is because for image processing, the aggregation and shuffle operations that are carried out during the reduce phase automatically are not required and only will affect the efficiency, i.e., longer time for job completion. However, this problem is overcome after integration with Hadoop Image Processing Interface (HIPI). HIPI is the image processing library that is designed to integrate with Hadoop MapReduce framework to perform image processing task. In our study, one of interesting findings is on the MapReduce framework streaming task that has failed for the numbers over the pre-defined threshold which causes broken pipe exception. Therefore, we propose the mapping processing in the MapReduce framework in solving writer identification problems.

With the integration of mapping module and reducing module into a single script, the output can be stored directly in a temporary directory without passing through the reduce phase. All the processes of sorting, shuffling and aggregating will be executed during reducing phase automatically provided by the MapReduce framework. Thus, the duration to complete all the jobs is reduced. Python is implemented to achieve multi-core processing for better performance. Mapping task is increased to separate the input data into smaller chunks to perform parallel processing. The result is good where the time taken to complete the whole process is lessening comparing to pre-optimizations.

5.4a Integration Analysis of MapReduce and Writer Documents

The integration of the proposed method is executed in the terminal after the HDFS server is started. The output directory, mapper file and extra commands are required to be specified. All outputs are stored in HDFS for further action. The user can ensure the job is running by accessing the Hadoop web UI to supervise and the terminal will display the MapReduce program is getting the input from HDFS. The percentages as shown in Fig. 14 show the jobs done in the mapping and reducing phases separately. When the jobs are completed, the percentage will give 100%.

```

huser@HadoopSlave1:~/hadoop$ hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar -D mapreduce.job.queue.name=default -D stream.non.zero.exit.is.failure=false -D mapreduce.
es=0 -input "words_2.hib" -mapper "python ./mapper.py" -output hadoop_tmp
17/06/04 16:04:22 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/06/04 16:04:22 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/06/04 16:04:22 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
17/06/04 16:04:23 INFO mapred.FileInputFormat: Total input paths to process : 1
17/06/04 16:04:23 INFO mapreduce.JobSubmitter: number of splits:1
17/06/04 16:04:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1873708218_0001
17/06/04 16:04:24 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/06/04 16:04:24 INFO mapred.LocalJobRunner: outputCommitter set in config null
17/06/04 16:04:24 INFO mapreduce.Job: Running job: job_local1873708218_0001
17/06/04 16:04:24 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
17/06/04 16:04:25 INFO mapred.LocalJobRunner: Waiting for map tasks
17/06/04 16:04:25 INFO mapred.LocalJobRunner: Starting task: attempt_local1873708218_0001_n_000000_0
17/06/04 16:04:25 INFO mapred.Task: Using ResourceCalculatorProcessTree : [ ]
17/06/04 16:04:25 INFO mapred.MapTask: Processing split: hdfs://localhost:54310/user/hduser/words_2.hib:0+922568
17/06/04 16:04:25 INFO mapred.MapTask: numReduceTasks: 0
17/06/04 16:04:25 INFO streaming.PipeMapRed: PipeMapRed exec [/usr/bin/python, ./mapper.py]
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.local.dir is deprecated. Instead, use mapreduce.task.id
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.local.dir is deprecated. Instead, use mapreduce.cluster.local.dir
17/06/04 16:04:25 INFO Configuration.deprecation: map.input.file is deprecated. Instead, use mapreduce.map.input.file
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.zip.on is deprecated. Instead, use mapreduce.job.skiprecords
17/06/04 16:04:25 INFO Configuration.deprecation: map.input.length is deprecated. Instead, use mapreduce.map.input.length
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.work.output.dir is deprecated. Instead, use mapreduce.task.output.dir
17/06/04 16:04:25 INFO Configuration.deprecation: map.input.start is deprecated. Instead, use mapreduce.map.input.start
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.job.id is deprecated. Instead, use mapreduce.job.id
17/06/04 16:04:25 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.job.user.name
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.task.is.map is deprecated. Instead, use mapreduce.task.ismap
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.task.id is deprecated. Instead, use mapreduce.task.attempt.id
17/06/04 16:04:25 INFO Configuration.deprecation: mapred.task.partition is deprecated. Instead, use mapreduce.task.partition
17/06/04 16:04:25 INFO streaming.PipeMapRed: R/W/S=1/0/0 in:NA [rec/s] out:NA [rec/s]
17/06/04 16:04:25 INFO streaming.PipeMapRed: R/W/S=10/0/0 in:NA [rec/s] out:NA [rec/s]
17/06/04 16:04:25 INFO streaming.PipeMapRed: R/W/S=100/0/0 in:NA [rec/s] out:NA [rec/s]
17/06/04 16:04:25 INFO mapreduce.Job: Job job_local1873708218_0001 running in user mode : false
17/06/04 16:04:25 INFO mapreduce.Job: map 0% reduce 0%
17/06/04 16:04:31 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hduser/words_2.hib:0+922568 > map
17/06/04 16:04:31 INFO mapreduce.Job: map 14% reduce 0%
17/06/04 16:04:52 INFO streaming.PipeMapRed: Records R/W=380/1
17/06/04 16:04:58 INFO mapred.LocalJobRunner: Records R/W=380/1 > map

```

Fig. 14: MapReduce program execution

When executing Spark MLib, the program must be able to create web UI and retrieve the data from HDFS as shown in Fig. 15. For this research, the input is split into 4 tasks because 1 core is assigned to each of the Spark's worker on the server. The user is able to keep track the running progress through the Spark web UI by accessing the URL that is displayed in the terminal instead of keeps tracking the progress displayed in the terminal. The final output from Spark is the identification rate and the total duration taken to complete the identification process. This represents that Spark has successfully completed the computation processes of all input data that are retrieved from HDFS. The integration is successful because of MapReduce program able to perform MapReduce jobs and stored the output in HDFS. Next, Spark program is able to retrieve the output data from MapReduce

program through HDFS to execute computation operation on the input data, and finally the output data from Spark program is displayed.

```

17/06/04 16:35:49 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.
17/06/04 16:35:49 INFO ut.SparkUI: Bound SparkUI to 172.17.0.1, and started at http://172.17.0.1:4040
17/06/04 16:35:49 INFO spark.SparkContext: Added JAR file:/home/hduser/hadoop/ar/target/scala-2.11/ar_2.11-1.0.jar at spark://172.17.0.1:38316/jars/ar_2.11-1.0.jar with timestamp 1496565349493
17/06/04 16:35:50 INFO scheduler.FairSchedulableBuilder: Created default pool default, schedulingMode: FIFO, minShare: 0, weight: 1
17/06/04 16:35:50 INFO executor.Executor: Starting executor ID driver on host localhost
17/06/04 16:35:50 INFO util.Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 36585.
17/06/04 16:35:50 INFO netty.NettyBlockTransferService: Server created on 172.17.0.1:36585
17/06/04 16:35:50 INFO storage.BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 172.17.0.1, 36585)
17/06/04 16:35:50 INFO storage.BlockManagerMasterEndpoint: Registering block manager 172.17.0.1:36585 with 3.6 GB RAM, BlockManagerId(driver, 172.17.0.1, 36585)
17/06/04 16:35:50 INFO storage.BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 172.17.0.1, 36585)
17/06/04 16:35:50 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@44ea608c{/metrics/json,null,AVAILABLE}
17/06/04 16:35:51 WARN spark.SparkContext: Use an existing SparkContext, some configuration may not take effect.
17/06/04 16:35:51 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@3af9a866{/SQL,null,AVAILABLE}
17/06/04 16:35:51 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@91c4a3ff{/SQL/json,null,AVAILABLE}
17/06/04 16:35:51 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@64d43929{/SQL/execution,null,AVAILABLE}
17/06/04 16:35:51 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@437ebf59{/SQL/execution/json,null,AVAILABLE}
17/06/04 16:35:51 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@1da6ee17{/static/sql,null,AVAILABLE}
17/06/04 16:35:51 INFO internal.SharedState: Warehouse path is 'file:/home/hduser/hadoop/ar/spark-warehouse'.
17/06/04 16:35:55 INFO memory.MemoryStore: Block broadcast_0 stored as values in memory (estimated size 62.1 KB, free 3.6 GB)
17/06/04 16:35:55 INFO memory.MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 28.6 KB, free 3.6 GB)
17/06/04 16:35:55 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on 172.17.0.1:36585 (size: 28.6 KB, free: 3.6 GB)
17/06/04 16:35:55 INFO spark.SparkContext: Created broadcast_0 from textFile at MLUtils.scala:99
17/06/04 16:35:56 INFO mapred.FileInputFormat: Total input paths to process : 1
17/06/04 16:35:56 INFO spark.SparkContext: Starting job: reduce at MLUtils.scala:92
17/06/04 16:35:56 INFO scheduler.DAGScheduler: Got job 0 (reduce at MLUtils.scala:92) with 18 output partitions
17/06/04 16:35:56 INFO scheduler.DAGScheduler: Final stage: ResultStage 0 (reduce at MLUtils.scala:92)
17/06/04 16:35:56 INFO scheduler.DAGScheduler: Parents of final stage: List()
17/06/04 16:35:56 INFO scheduler.DAGScheduler: Missing parents: List()
17/06/04 16:35:56 INFO scheduler.DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[S] at map at MLUtils.scala:90), which has no missing parents
17/06/04 16:35:56 INFO memory.MemoryStore: Block broadcast_1 stored as values in memory (estimated size 3.0 KB, free 3.6 GB)
17/06/04 16:35:56 INFO memory.MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 2.1 KB, free 3.6 GB)
17/06/04 16:35:56 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on 172.17.0.1:36585 (size: 2.1 KB, free: 3.6 GB)
17/06/04 16:35:56 INFO spark.SparkContext: Created broadcast_1 from broadcast at DAGScheduler.scala:1012
17/06/04 16:35:56 INFO scheduler.DAGScheduler: Submitting 18 missing tasks from ResultStage 0 (MapPartitionsRDD[S] at map at MLUtils.scala:90)
17/06/04 16:35:56 INFO scheduler.TaskSchedulerImpl: Adding task set 0.0 with 18 tasks
17/06/04 16:35:56 INFO scheduler.FairSchedulableBuilder: Added task set TaskSet_0 tasks to pool default
17/06/04 16:35:56 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localhost, partition 0, PROCESS_LOCAL, 5436 bytes)
17/06/04 16:35:56 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, localhost, partition 1, PROCESS_LOCAL, 5436 bytes)
17/06/04 16:35:56 INFO scheduler.TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, localhost, partition 2, PROCESS_LOCAL, 5436 bytes)
17/06/04 16:35:56 INFO scheduler.TaskSetManager: Starting task 3.0 in stage 0.0 (TID 3, localhost, partition 3, PROCESS_LOCAL, 5436 bytes)
17/06/04 16:35:56 INFO executor.Executor: Running task 2.0 in stage 0.0 (TID 2)
17/06/04 16:35:56 INFO executor.Executor: Running task 0.0 in stage 0.0 (TID 0)
17/06/04 16:35:56 INFO executor.Executor: Running task 1.0 in stage 0.0 (TID 1)
17/06/04 16:35:56 INFO executor.Executor: Running task 3.0 in stage 0.0 (TID 3)
17/06/04 16:35:56 INFO executor.Executor: Fetching spark://172.17.0.1:38316/jars/ar_2.11-1.0.jar with timestamp 1496565349493
17/06/04 16:35:56 INFO client.TransportClientFactory: Successfully created connection to /172.17.0.1:38316 after 28 ms (0 ms spent in bootstraps)
17/06/04 16:35:56 INFO util.Utils: Fetching spark://172.17.0.1:38316/jars/ar_2.11-1.0.jar to /tmp/spark-24749338-f3cd-40a3-b0bc-c3dd04f8dc21/userFiles-7dd6abc4-f18c-4642-a90d-b5bd73c0c5b9/fetch
2265316050413774.tmp
17/06/04 16:35:56 INFO executor.Executor: Adding file:/tmp/spark-24749338-f3cd-40a3-b0bc-c3dd04f8dc21/userFiles-7dd6abc4-f18c-4642-a90d-b5bd73c0c5b9/ar_2.11-1.0.jar to class loader
17/06/04 16:35:56 INFO rdd.HadoopRDD: Input split: hdfs://localhost:54310/user/hduser/train.txt:402653184+134217728
17/06/04 16:35:56 INFO rdd.HadoopRDD: Input split: hdfs://localhost:54310/user/hduser/train.txt:268435456+134217728
17/06/04 16:35:56 INFO rdd.HadoopRDD: Input split: hdfs://localhost:54310/user/hduser/train.txt:0+134217728
17/06/04 16:35:56 INFO rdd.HadoopRDD: Input split: hdfs://localhost:54310/user/hduser/train.txt:134217728+134217728

```

Fig. 15: Spark's execution

6 Conclusion and Discussions

In this study, we found that Hadoop MapReduce is a promising framework for processing Big Data due to its friendliness in configuring the parameters. It is also user – friendly since the user needs to write two functions: mapping function and reducing function from a large amount of data extraction which is performed in

parallel. Fault tolerance will be provided by the framework where the user does not need to worry about the loss of data if the error occurs.

All the data will be separated into many chunks before passing to the processing node, and data replication will be executed to back up all data. Spark is proven for high processing speed due to RAM computation, but insufficient storage for post-processing. However, this problem is solved by integrating with Hadoop. The integration of Hadoop and Spark can utilize the advantage of these two technologies which is the processing speed of Spark and the storage that is provided by Hadoop (HDFS).

In this paper, we implement Authorship identification using multilayer perceptron classifier in Spark's MLlib for classification purpose. However, further improvements are required to obtain better identification rate by converting all the data into the data format that is accepted by Spark application without losing writers' information. Furthermore, image processing library is required to achieve better identification which is currently not available in Spark MLlib.

ACKNOWLEDGEMENTS

The authors would like to thank Ministry of Higher Education (MOHE) and Universiti Teknologi Malaysia (UTM) for their support in Research and Development. This work is partially supported by the UTM Research University Grant Scheme FRGS (4F786 & 4F802) and RUG (17H62).

References

- [1] Wu, X., Zhu, X., Wu, G. Q., & Ding, W. (2014). Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1), 97-107.
- [2] Narayanan, A., Paskov, H., Gong, N. Z., Bethencourt, J., Stefanov, E., Shin, E. C. R., & Song, D. (2012, May). On the feasibility of internet-scale author identification. In *Security and Privacy (SP), 2012 IEEE Symposium on* (pp. 300-314). IEEE.
- [3] Landset, S., Khoshgoftaar, T. M., Richter, A. N., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(1), 24.
- [4] Costantini, L., & Nicolussi, R. (2015). Performances evaluation of a novel Hadoop and Spark based system of image retrieval for huge collections. *Advances in Multimedia*, 2015, 11.

- [5] Tan, R. H. R., & Tsai, F. S. (2010, October). Authorship identification for online text. In *Cyberworlds (CW), 2010 International Conference on* (pp. 155-162). IEEE.
- [6] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107-113.
- [7] Jiang, W., Ravi, V. T., & Agrawal, G. (2010, May). A map-reduce system with an alternate api for multi-core environments. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (pp. 84-93). IEEE Computer Society.
- [8] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S. H., Qiu, J., & Fox, G. (2010, June). Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM international symposium on high performance distributed computing* (pp. 810-818). ACM.
- [9] Ghazi, M. R., & Gangodkar, D. (2015). Hadoop, MapReduce and HDFS: a developers perspective. *Procedia Computer Science*, 48, 45-50.
- [10] Marinai, S., Gori, M., & Soda, G. (2005). Artificial neural networks for document analysis and identification. *IEEE Transactions on pattern analysis and machine intelligence*, 27(1), 23-35.
- [11] Hiremath, P. S., Shivashankar, S., Pujari, J. D., & Kartik, R. K. (2010, December). Writer identification in a handwritten document image using texture features. In *Signal and Image Processing (ICSIP), 2010 International Conference on* (pp. 139-142). IEEE.
- [12] Mohammed, B. O., & Shamsuddin, S. M. (2011). Feature discretization for individuality representation in twins handwritten identification. *Journal of Computer Science*, 7(7), 1080.
- [13] Zamora-Martinez, F., Frinken, V., España-Boquera, S., Castro-Bleda, M. J., Fischer, A., & Bunke, H. (2014). Neural network language models for off-line handwriting identification. *Pattern Identification*, 47(4), 1642-1652.
- [14] *HIPI - Hadoop Image Processing Interface :: Introduction*. (2017). *Hipi.cs.virginia.edu*. Retrieved 29 June 2017, from <http://hipi.cs.virginia.edu/>
- [15] *Overview - Spark 2.0.0 Documentation*. (2017). *Spark.apache.org*. Retrieved 29 June 2017, from <https://spark.apache.org/docs/preview/>
- [16] *Welcome to opencv documentation! — OpenCV 2.4.13.2 documentation*. (2017). *Docs.opencv.org*. Retrieved 29 June 2017, from <http://docs.opencv.org/2.4.13.2/>