PAVE: Write-print Creation with MapReduce

Leo St. Amour, Frederick Ulrich, Andreas Kellas, Alexander Molnar, and Suzanne J. Matthews

Abstract—Cyber-crime is becoming alarmingly common through the use of anonymous e-mails. Author attribution helps digital forensics investigators filter through a large set of possible authors and focus traditional investigative techniques on the most probable culprits. A recent promising technique is to construct a write-print for each known author and compare it to the write-print extracted from the anonymous message(s). A write-print is a unique digital fingerprint created by mining frequent patterns from a particular author's writing style. Parallel computing enables us to leverage multiple cores in the creation of author write-prints. We introduce Parallel Author Verification of E-mail (PAVE), a MapReduce algorithm for generating author write-prints in parallel. Our algorithm is able to achieve up to 90% accuracy when tested on a subset of the Enron dataset. We believe the community will find the PAVE system useful to expedite author identification in time sensitive situations.

Index Terms—author verification, parallel computing, data mining, write-print, Enron

1 Introduction

The author identification problem seeks to determine the most likely author from a group of known authors. Author attribution methods have traditionally been applied to the literary domain. Famous examples include determining the authorship of the Federalist Papers [1], and the recent outing of JK Rowling [2] as the author of *The Cuckoo's Calling*.

The advent of the digital age caused the author identification problem to extend beyond literary authorship to include electronic authorship of documents such as e-mails and blog posts. Author verification is especially difficult for e-mails, as the length of e-mails tend to be short. Longer documents are easier to analyze, due to the presence of more data. Forsyth and Holmes [3] suggest that 250 words is the minimum length for effective authorship attribution. A more recent survey suggests that it is not yet possible to determine a minimum threshold [4]. Despite this, accurate author attribution on short segments of text becomes more desirable as the popularity of short electronic communications such as e-mails and texting continues to grow.

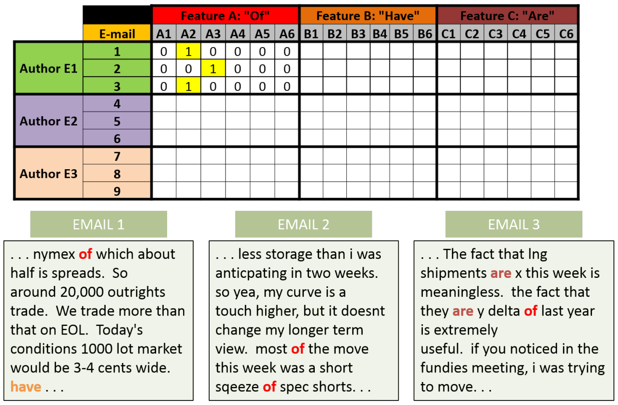
Write-print analysis [5], [6], [7] is a recent e-mail author attribution technique that shows considerable promise. To create a write-print, a combination of stylometric features is used to identify a set of frequent patterns that represents an author's writing style [5]. In typical author attribution approaches, these features are selected individually for analysis. Selecting "good" features that capture an author's writing style is critical to generating useful write-prints. Adding additional features at random can inadvertently increase the amount of noise in the dataset, reducing accuracy [7]. Prior work [5] for feature selection conducts hueristic searches on a feature space. As the feature space gets large, this step can take a long time.

One of our major goals is to enable author identification in time-sensitive situations. For this reason, we chose to focus on the write-print method implemented by Iqbal *et. al.* in their AuthorMiner software. AuthorMiner is novel because it filters out features that are shared among multiple authors. Thus, each author's write-print is composed only of the features that are unique to them. This eliminates the feature selection step. When run on a subset of the Enron e-mail corpus [8] consisting of 6 authors each with 20 e-mails (120 e-mails), AuthorMiner achieves an attribution accuracy of between 86% to 90% [7]. On a different configuration of 10 authors with 10 e-mails (100 e-mails), attribution accuracy ranges from 80% to 90% [7]. These results suggest that AuthorMiner's write-print method is an effective technique for e-mail author attribution.

We focus on reproducing the results achieved by AuthorMiner. The AuthorMiner code (like other systems for e-mail author attribution) is not publicly available. The paper does not report running time, leading us to hypothesize the method may be inefficient. Furthermore, the authors only test their approach on a maximum of 10 authors and 120 e-mails. We wish to explore if the reason for this restriction was due to runtime, accuracy, or a combination of both. To this end, we developed a MapReduce algorithm called Parallel Author Verification of E-mail (PAVE) for determining author write-prints in parallel. Our goal was to produce write-prints quickly and verify the accuracy of write-prints on the data sizes specified in the AuthorMiner paper.

Lastly, while much research exists on techniques for e-mail author attribution, few (if any) practical systems exist. Those that do are closed-source, meaning that it is

- *Dr. Matthews is with the Department of Electrical Enginering and Computer Science, United States Military Academy, West Point, NY. E-mail: suzanne.matthews@usma.edu*
- *2LTs St. Amour and Ulrich are members of the Army Cyber branch, MIT Lincoln Lab Military fellows, and graduate students at Northeastern University, Boston, MA.*
- *2LTs Kellas and Molnar are members of the Army Cyber branch and are currently completing BOLC training at Fort Gordon, GA.*



		Feature A: "Of"						Feature B: "Have"						Feature C: "Are"					
	E-mail	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	C4	C5	C6
Author E1	1	0	1	0	0	0	0												
	2	0	0	1	0	0	0												
	3	0	1	0	0	0	0												
Author E2	4																		
	5																		
	6																		
Author E3	7																		
	8																		
	9																		

EMAIL 1

. . . nymex of which about half is spreads. So around 20,000 outrights trade. We trade more than that on EOL. Today's conditions 1000 lot market would be 3-4 cents wide. have . . .

EMAIL 2

. . . less storage than i was anticpating in two weeks. so yea, my curve is a touch higher, but it doesnt change my longer term view. most of the move this week was a short sqeeze of spec shorts. . .

EMAIL 3

. . . The fact that lng shipments are x this week is meaningless. the fact that they are y delta of last year is extremely useful. if you noticed in the fundies meeting, i was trying to move. . .

Fig. 1. Feature Identification

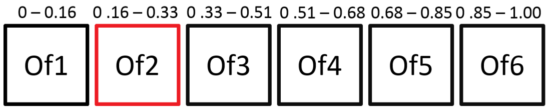
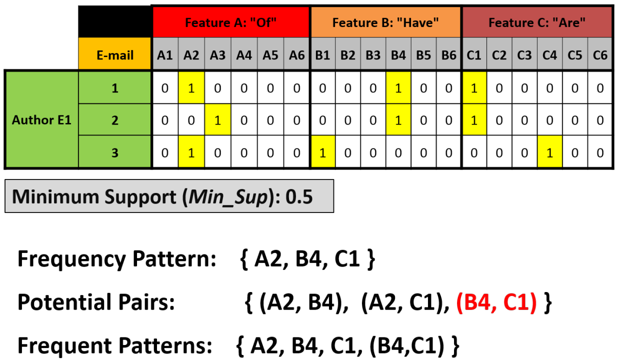


Fig. 2. Equal-width Discretized Binning



		Feature A: "Of"						Feature B: "Have"						Feature C: "Are"					
	E-mail	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	C4	C5	C6
Author E1	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
	2	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
	3	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0

Minimum Support (*Min_Sup*): 0.5

Frequency Pattern: { A2, B4, C1 }

Potential Pairs: { (A2, B4), (A2, C1), (B4, C1) }

Frequent Patterns: { A2, B4, C1, (B4,C1) }

Fig. 3. Extracting frequent patterns

almost impossible for an organization to quickly acquire the tools necessary to enable e-mail author attribution (unless set up ahead of time) in cases where threats of a time-sensitive nature are made. Unlike previous implementations, we plan to make the final version of PAVE available to the general public.

The rest of this paper is organized as follows. Section 2 summarizes AuthorMiner's core algorithm. We cover our MapReduce algorithm in Section 3. In Section 4 and Section 5 we discuss our experimental framework and results. Finally, we conclude in Section 6.

2 AUTHORMINER

Iqbal *et. al.* present a novel algorithm and system for determining e-mail authorship called AuthorMiner. The algorithm can be broken into three phases: mining a candidate set of features; computing frequent patterns; and filtering out common frequent patterns to compute the write-print. We describe the different phases of the algorithm in the subsections below. To illustrate the workings of AuthorMiner, we use an example consisting of three authors, each with three emails ($n = 3$, $m = 3$).

2.1 Phase 1: Mining a Candidate Set of Features

The first phase concentrates on extracting features of interest from an author's email set and determining which features are considered frequent. Suppose we want to mine three authors for features A, B, and C, which represent words "of" ,"have", and "are", respectively. Figure 1 demonstrates the results of extracting feature A from author $E1$'s three emails, which are shown at the bottom of the figure as snippets. The frequency of each feature is normalized by the length of the e-mail.

AuthorMiner uses equal-width discretization as the primary binning strategy for each feature. Figure 2 demonstrates this process. Consider the word "of", which appears three times in e-mail e_1. We normalize this frequency by the number of elements in the feature class (e.g. function words) Since e-mail e_1 has 11 total function words, the relative frequency of the word "of" is 0.27, placing the word in the second bin, which covers frequency range 0.16 to 0.33. This is represented as feature $A1$. Author $E1$'s uses of the word "of" fell into the second bin for Emails e_1 and e_3, and into the third bin for Email e_2. This process continues until each feature is placed into the appropriate bin for all emails.

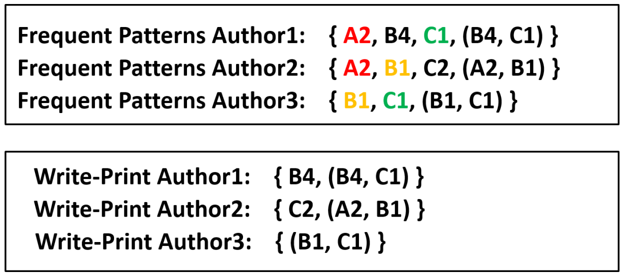
2.2 Phase 2: Computing Frequent Patterns

A feature is considered *frequent* if it appears within a specified percentage called the minimum support (Min_Sup) of an author's e-mails. In the examples that follow, we use a $Min\_Sup = 0.5$, indicating that a feature must appear in at least 50% of the author's e-mails in order to be considered frequent. At the end of the last phase, we calculate an initial set of features by considering only those features with a frequency greater than the Min_Sup. In our example, the features $A2$, $B4$, $C1$ meet the threshold $Min\_Sup = 0.5$ and constitute the author's initial frequent pattern set, or candidate set.

Next, we look at all n-combinations of elements in the candidate set to identify which of those meet the Min_Sup threshold, where $n > 1$. AuthorMiner uses the *a-priori* combinatorial feature mining algorithm [9] to accomplish this. Returning to our example in Figure 3, where we concentrate on author $E1$ and his set of e-mails, the candidate combinations are ($A2$, $B4$), ($A2$, $C1$), and ($B4$, $C1$). The pair ($B4$, $C1$) is the only one to meet the threshold of $Min\_Sup = 0.5$ because these features appear together in the same e-mail in more than half of the e-mails. Thus, author $E1$'s set of frequent patterns is $\{A2, B4, C1, (B4, C1)\}$.

2.3 Phase 3: Computing Write-prints

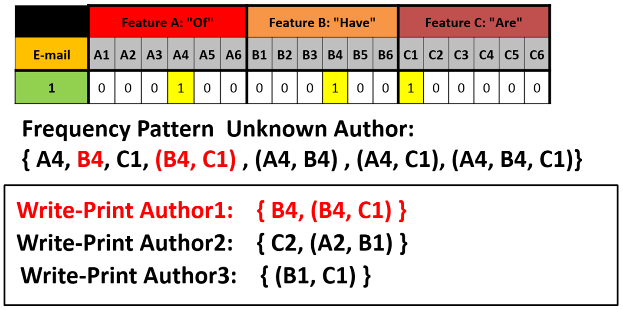
Phase 3 of AuthorMiner involves filtering out frequent patterns that are common to more than one author. Figure 4 shows the frequent pattern sets for our three authors. Note that features $A2$, $B1$, and $C1$ are present



Frequent Patterns Author1: { A2, B4, C1, (B4, C1) }
Frequent Patterns Author2: { A2, B1, C2, (A2, B1) }
Frequent Patterns Author3: { B1, C1, (B1, C1) }

Write-Print Author1: { B4, (B4, C1) }
Write-Print Author2: { C2, (A2, B1) }
Write-Print Author3: { (B1, C1) }

Fig. 4. Eliminating Common Frequent Patterns to Generate Write-prints



	Feature A: "Of"						Feature B: "Have"						Feature C: "Are"					
E-mail	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	C4	C5	C6
1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0

Frequency Pattern Unknown Author:
{ A4, B4, C1, (B4, C1) , (A4, B4) , (A4, C1), (A4, B4, C1)}

Write-Print Author1: { B4, (B4, C1) }
Write-Print Author2: { C2, (A2, B1) }
Write-Print Author3: { (B1, C1) }

Fig. 5. Comparing write-prints of known authors to unknown e-mail

in more than one author's frequent pattern set, and therefore must be removed. Once all common frequent patterns are eliminated, the remaining set of patterns will be unique for each author. Together, these unique frequent patterns form a write-print (WP). The write-prints for the three authors can also be seen in Figure 4. For example, for the first author, the elimination of frequent patterns $A2$ and $C1$ yields a write-print consisting of the pattern set $\{B4, (B4, C1)\}$. AuthorMiner's creators note that this portion of AuthorMiner can take exponential time with respect to the total number of frequent patterns and the number of authors. Their exponential algorithm takes $O(|FP(E)|^n)$ time, where $|FP(E)|$ is the number of frequent patterns belonging to author E and n is the number of authors under consideration.

2.4 Comparing Write-Prints

To predict the author of an unknown e-mail, the write-print of each author is compared to the unknown e-mail. AuthorMiner determines similarity by assigning each author a score based on Equation 1 [7]:

$$Score(u \approx WP(E_i)) = \frac{\sum_{j=1}^{p} support(MP_j|E_i)}{|WP(E_i)|}$$

The score for each author i (E_i) is calculated by summing up the support (or relative frequency) of each pattern that is common between E_i' s write-print and the unknown email and dividing it by the number of features contained in the write-print. Higher scores indicate greater similarity. Figure 5 illustrates the three write-prints that were created earlier being compared to an unknown email. The frequent pattern B4, and the pair (B4, C1) appear in both the unknown email and in author $E1$'s write-print. In this particular case, none of the other features are in common between the unknown email and either authors $E2$ and $E3$. This means the unknown author's writing style most closely matches that of author $E1$, and this is returned as the predicted author.

2.5 Limitations

There are several key limitations to AuthorMiner's approach. Chief among these is the exponential complexity of the step required to eliminating common frequent patterns: $O(|FP(E)|^n)$ [7]. We suspect this is why AuthorMiner's experimentation was restricted to relatively small numbers of authors ($n \leq 10$) and e-mails per author ($m \leq 20$). The algorithm also only uses short words and function words. We were curious how additional features would impact performance.

3 The PAVE Algorithm

PAVE employs the MapReduce [10] framework for generating write-prints in parallel. Popularized by Google, MapReduce is designed to parallelize the computations across clusters containing large numbers of machines [10]. To utilize MapReduce, programmers are required to implement two functions: `map()` and `reduce()`. The underlying scheduling framework automates the rest of the process. The `map()` function processes the input and produces an intermediate set of *(key, value)* pairs. The `reduce()` function combines and processes all values with the same key to produce some reduced output. We utilize Phoenix++ [11], an open-source, shared-memory implementation of MapReduce that enables users to write high performance code [12]. We hybridize Phoenix++ with MPI to exploit multiple nodes and cores. The PAVE algorithm is split into three phases, mimicking AuthorMiner's setup.

In the subsections below, we use the same $n = 3$ and $m = 3$ example from the AuthorMiner section. The MPI layer starts by requesting n nodes, and running Phase 1 MapReduce on each. This first MapReduce phase processes the author's m e-mails and extracts a candidate set of frequent patterns for each author. Thus, for Phase 1, there are n MapReduce jobs running in parallel (multi-node), with each MapReduce job processing m e-mails in parallel (multi-core). In Phase 2, we use the *a-priori* algorithm to generate frequent patterns for each author, a serial process. In Phase 3, we utilize another MapReduce phase (multi-core) to eliminate common frequent patterns and compute author write-prints. We describe each these phases in detail below.

3.1 Phase 1: Extract Frequent Patterns

Our first MapReduce phase mines an author's set of e-mails for selected features and computes the candidate

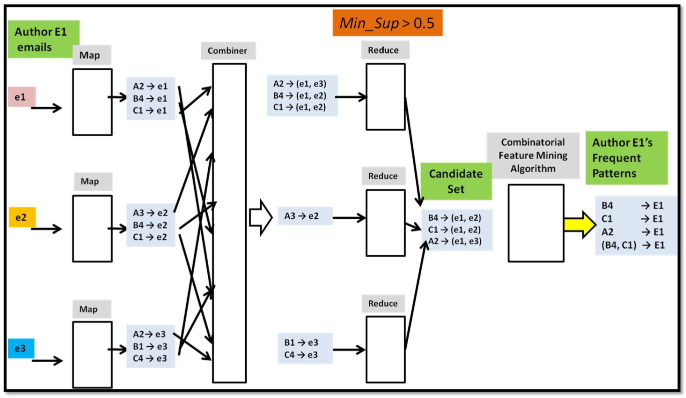


Fig. 6. First MapReduce Phase: Extract Frequent Patterns

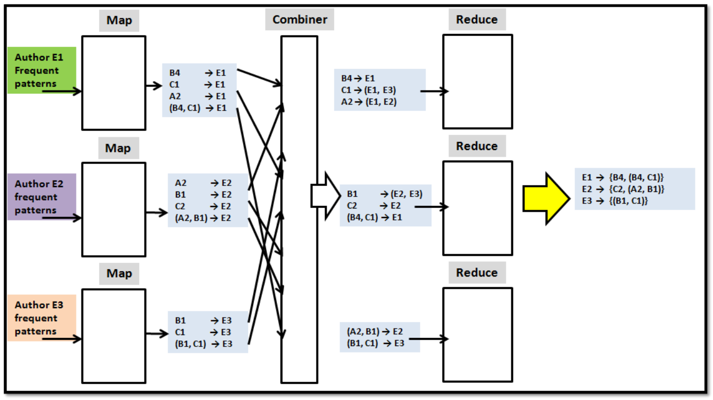


Fig. 7. Second MapReduce Phase: Computing Write-prints

set (1-frequent patterns) for that author. The feature sets we consider are short words and function words. In order to reproduce AuthorMiner's results, we use the same list [13] of function words used by AuthorMiner, and define a short word as a word with four or less characters. Each author is assigned to a separate node via an MPI wrapper. The wrapper then executes the described MapReduce phase for each author on its assigned node's multiple cores.

Figure 6 demonstrates this MapReduce process. In our example, author E_1 has written three e-mails: e_1, e_2, and e_3. Each mapper extracts the set of features of interest contained in a particular e-mail, and emits intermediate *(key, value)* pairs of the feature and e-mail id. The features of interest are represented by A, B, and C (corresponding to "of", "have" and "are). In accordance with AuthorMiner, PAVE uses equal-width discretized binning of each feature, and a total of six bins. Thus $A2$ represents that "of" falls into bin 2. E-mail e_1 emits three features, $A2$, $B4$, and $C1$. Each of these features is emitted along with the id of the e-mail it came from (in this case, e_1) to form an intermediate pair.

The intermediate pairs are combined and sorted into *(key, list(values))* pairs, where each unique feature id is paired with the associated set of e-mail ids that contain it. These pairs are input into the reduce function. The reduce function only emits pairs where the feature occurs above a minimum threshold (*Min_Sup*). In this example, the $Min\_Sup = 0.5$, representing that each feature must appear in at least 50% of the author's e-mails. The only features that satisfy this requirement are $A2$, $B4$, and $C1$. These three features form the 1-pattern set of frequent patterns for author $E1$. Note that each independent reducer operates in parallel and on multiple cores.

3.2 Phase 2: Computing Frequent Patterns

We use the trie-based *a-priori* algorithm described by Bodon [14], which is open-sourced at http://www.cs.bme.hu/~bodon/en/apriori/. Bodon's *a-priori* algorithm is implemented in C++, aiding in our ability to incorporating it into PAVE. While multiple papers discuss parallel *a-priori* algorithms, we were unable to find an open-source implementation. As a result, the *a-priori* portion of the PAVE system is serial. Designing and implementing a parallel *a-priori* implementation was unfortunately outside the scope of the current work.

3.3 Phase 3: Computing Write-prints

The next phase of MapReduce identifies the set of frequent patterns unique to each author. Unlike the previous MapReduce phase, which operate on multiple nodes and multiple cores, this phase operates on multiple cores of a single node. This layout is necessary in order to compare each author. The input to this phase is the frequent patterns of authors E_1, E_2, and E_3. Following our example, author $E1$'s frequent patterns are $\{A2, B4, C1, (B4, C1)\}$, author $E2$'s frequent patterns are $\{A2, B1, C2, (A2, B1)\}$, and author $E3$'s frequent patterns are $\{B1, C1, (B1, C1)\}$.

Each mapper emits a *(key, value)* pair consisting of a frequent pattern feature and its corresponding author. For example, for author $E1$, we emit the pairs ($A2$, $E1$), ($B4$, $E1$), ($C1$, $E1$), and ($\{B4, C1\}$, $E1$). The combiner identifies the set of authors common to a particular frequent pattern feature, and outputs *(key,list(value))* pairs, where the frequent pattern is the key, and the list of values is the set of authors containing that particular frequent pattern. For example, the combiner emits ($C1$, ($E1$, $E2$)), since $C1$ is a frequent pattern of both authors $E1$ and $E2$. The reduce function then emits frequent patterns that are unique to a particular author. Thus, frequent pattern $C1$ is not emitted, since it shared by authors $E1$ and $E2$.

The emitted frequent patterns are organized to form write-prints for each author. We follow AuthorMiner's example for comparing unknown e-mails to the known write-prints (Figure 5).

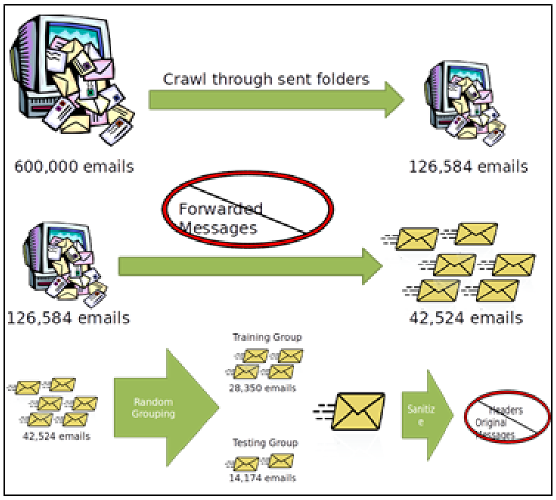


Fig. 8. Overview of Data Preprocessing

4 Experimental Setup

In this section we outline our experimental methodology and setup. We will discuss how we pre-process our email data set, set up our experiments, and discuss our methodology for predicting the author of an unknown set of emails. While we are attempting to replicate the results of AuthorMiner, we have made a number of assumptions that we will discuss in this section.

4.1 Data Pre-processing

The Enron Corpus [8] is the largest publicly available database of real e-mails. It contains over $600,000$ emails written by 158 Enron employees [8]. The emails are organized into 158 directories, one corresponding to each author. Each author's directory contains separate directories for the inbox, sent folder, deleted items, and others.

AuthorMiner's creators do not explicitly state how the Enron corpus was pre-processed prior to their experimentation. For the purposes of our research, we decide to only use emails that are in "sent" directories, as these e-mails are almost always written by the author in question. This reduces the dataset to $126,584$ e-mails. For each author, all forwarded emails are removed, and headers and reply chains are stripped, leaving only author-written content (Figure 8). After pre-processing, our data set consists of $42,524$ e-mails. The median number of e-mails per author is 289.

Following Iqbal's example, we divide our data set into two sub-sets. Two-thirds of each author's e-mails ($28,350$ total) form the training dataset and are dedicated to creating the reference write-print. The last one-third of e-mails ($14,174$) of each author are used for testing, and are used to create a write-print for our "unknown" author. To reproduce AuthorMiner's results (with $m = 20$ e-mails for the reference write-print), we need to use authors with a minimum of 30 e-mails. Of the 150 authors, only 119 (81%) had 30 or more e-mails.

There are limits in the experimental design of AuthorMiner – for example, the authors do not report the rates of true positives or false negatives. However, our goal in conducting this work is to reproduce AuthorMiner's results. To that end, we did our best not to stray from the experimental design as described in the paper, and make assumption only as necessary.

4.2 Benchmarking Details

Performance benchmarking is conducted on the Garnet high performance computing cluster hosted by the Department of Defense's Engineer Research and Development Center (ERDC), which consists of $4,716$ total compute nodes. Each compute node has 64GB of RAM and 32 AMD Interlagos Opteron cores operating at a clock rate of 2.5GHz. Since our primary goal is to reproduce AuthorMiner's experimental setup, we initially limit our experimentation to at most 6 nodes.

We measure the overall run-time of our system. Phases 1 and 2 are tested as separate components for speedup, as each has separate parallelizable components. Phase 1 is tested in both single node mode and multiple node mode, in which each author's frequent patterns were mined on a separate node, while Phase 2 was tested only in single node format.

We conducted our experiments by randomly selecting n authors from our training dataset and m of those author's e-mails. In order to reproduce the AuthorMiner results, we set $n = 6$ and $m = 20$ in our experiments. Of the n authors, one is randomly selected to serve as the "unknown" author. That author's testing e-mails are used to form an unknown write-print, which is then compared to the write-prints created from the training sets of our n authors. To replicate AuthorMiner's results, we also restrict the Min_Sup values to 0.2, as this support yielded the high accuracy numbers for AuthorMiner. We use 6 bins for our equal-width discretized binning, as this yielded the best results [7] for AuthorMiner.

To try and replicate AuthorMiner's results, we initially restrict the feature classes to include just function and short words. We also add an additional feature (bi-words), and study how it affects accuracy. Finally, we create three distinct subsets of the Enron dataset. Non-selective (containing no word minimum), a fifty-word minimum set (each e-mail contains at least 50 words), and a hundred-word minimum set (each e-mail contains at least 100 words). We present run-time and accuracy results for each of these classes.

5 Results

We perform extensive testing to assess both PAVE's accuracy and speed. Our initial testing was hindered by run time. The serial *a-priori* dominated the run-time, especially when a large number of frequent patterns were outputted. Our solution was to implement a threshold

Method	Top 1	Top 2
Relative	33.9%	57.6%
Normal	30.0%	50.0%
Tight (.3)	25.8%	36.2%
Tight (.1)	23.3%	30.0%

TABLE 1
Binning method accuracy

Cores	MapReduce	A-Priori	Trans	I/O	Total
1	0.991	157.137	0.008	140.503	298.64
2	0.517	134.118	0.008	132.523	267.167
4	0.235	140.361	0.009	146.676	287.281
8	0.085	140.301	0.008	144.801	285.195
16	0.136	139.57	0.007	145.088	284.801
32	0.119	139.885	0.009	144.937	284.951

TABLE 2
Phase 1 and 2 (One node): Six authors, 20 e-mails (in seconds)

Cores	MapReduce
1	0.484
2	0.416
4	0.260
8	0.406
16	0.339
32	0.472

TABLE 3
Phase 1 (multi-node): Six authors, 20 e-mails (in seconds)

Cores	MapReduce
1	9.120
2	7.477
4	6.643
8	6.550
16	5.917
32	6.317

TABLE 4
Phase 3: Six authors, 20 e-mails (in seconds)

on the *a-priori* algorithm, which limited the number of n-frequent patterns. So, if $n = 4$, the testing for additional frequent patterns would stop at 4-frequent patterns. This limit significantly reduced our run-time bottleneck. For speed, the *a-priori* threshold was set to 4.

After determining an *a-priori* threshold that would strike a balance between accuracy and speed, we tested the impact of our different binning methods. The results from our binning experiments can be seen in Table 1. The equal-width discretized binning strategy described in previous sections resulted in a large number of features falling into the first bin, which affected accuracy. We tried a number of related binning strategies, including discretizing from the range $0 \ldots 0.1$ and placing those higher in bin 6 (tight 0.1), and doing something similar with discretization occuring between range $0 \ldots 0.3$ (tight 0.3). We also tested a "relative" binning strategy, which determined the "min" frequency and the "max" frequency relative to a feature class, and performed binning based on this min and max. Our experimentation suggests that the "relative" binning method outperforms both the "tight" implementation and the normal equal-width discretization, leading us to adopt it as our binning method of choice in the following experiments.

5.1 Run-time benchmarking

We measure the performance of Phases 1, 2 and 3 separately, and consider the performance of the system as a whole. Phase 1 was tested in both single node mode and multiple node mode, in which each author's frequent patterns were mined on a separate node, while Phase 2 was tested only in single node format.

Table 2 depicts the the combined run-time results of Phase 1 and 2 of our analysis on one-node. Phase 1 is the parallel MapReduce portion. Phase 2 is the serial *a-priori* algorithm. The MapReduce portion of the algorithm is very fast, processing six authors and 20-emails in 0.085 seconds on 8 cores. This corresponds to a 11.65 times speedup over the 1-core execution. Despite the fast execution of the MapReduce component, the serial *a-priori* execution dominates execution time. For a $Min\_Sup = 0.2$, a large number of frequent patterns are generated. The larger the number of generated frequent patterns, the longer the *a-priori* time, and the longer the I/O time required to write the frequent patterns to a file. Table 3 shows only the run-time of our Phase 1 MapReduce when using multiple nodes. In this case, there is so little data going to the multiple nodes that the communication overhead of using MPI outstrips the benefit of using multiple nodes. We predict that with larger number of authors and e-mails, this will become less of an issue.

Table 4 shows the performance of the Phase 3 MapReduce run. Recall the purpose of this phase was to eliminate common frequent patterns. Even on a single core, our approach is fast, eliminating the common frequent patterns in a group of 6 authors in 9.120 seconds. As we increase the number of cores, we see a decrease in run-time through 16 cores. Phase 3 MapReduce can eliminate common frequent patterns in 5.917 seconds on 16 cores, representing a speedup of 54%.

It takes 5 minutes for our parallel implementation to generate the write-prints of 6 authors and a total of 120 e-mails. While the serial *a-priori* program takes up the majority of our run-time, our MapReduce components take just seconds to execute.

5.2 Accuracy benchmarking

The goal of our research was to reproduce the results presented in the AuthorMiner paper, which reported accuracy results of up to 90% [7]. To attempt to reproduce these results, we conducted a number of experiments. After determining which parameters yielded the highest accuracy, we were still not able to duplicate the high

Accuracy Results (n=6, m=20, bins=6, minsup=.2, btype=relative, ap_threshold=4, features=all)		
	Top 1	Top 2
Non-selective	52.7%	65.5%
50 word minimum	81.6%	91.2 %
100 word minimum	90.9%	93.1%

TABLE 5
Accuracy results on different subsets of Enron data set.

accuracy rates reported by AuthorMiner. However, we note that Iqbal *et. al* were not explicit about what subset of the Enron database they used. We therefore created two subsets: a fifty word minimum and a hundred-word minimum, where each set included authors and their e-mails that had at least fifty or a hundred words, respectively.

Running PAVE on these new datasets yielded much higher accuracy. Our results are shown in Table 5. Before setting a minimum number of words per e-mail, our algorithm averaged 52.7% Top 1 accuracy. Enforcing a 50-word minimum resulted in a Top 1 accuracy of 81.6%. Further increasing the minimum to 100 words allowed PAVE to achieve accuracy of 90.9%. We believe that by eliminating e-mails with smaller word counts, we are increasing the number of possible features that can be considered unique for an author. This leads us to conclude that e-mails containing a single word or phrase such as "Thanks" or "sounds good" do not greatly contribute to an author's write print, and should be removed from consideration.

6 Conclusion

In this paper we introduce PAVE, a parallel algorithm for author verification of e-mail. PAVE is a parallelization of the AuthorMiner algorithm, proposed by Iqbal *et. al.* in 2008. The original AuthorMiner algorithm was tested on a subset of 120 e-mails of the Enron corpus. We hypothesized that the reason for this small subset was the exponential run-time of elimination of frequent patterns. Despite this, the AuthorMiner algorithm reportedly enjoyed very high accuracy results. Our initial goal in creating PAVE was to create a fast algorithm capable of doing authorship in time-sensitive situations, and reproduce the results presented by the creators of AuthorMiner. To this end, we designed a shared memory distributed MapReduce algorithm that we deployed on the DOD ERDC Garnet cluster. Using MapReduce allows us to eliminate frequent patterns in a matter of seconds.

While these results highlight the efficiency of our implementation, it raises several questions. First, how fast does each of the components of AuthorMiner run? AuthorMiner's authors did not publish either their implementation or run-time numbers. The *a-priori* step took the majority of the time, constituting over 99% of the total running time. It is possible that AuthorMiner implements a more efficient (yet unavailable) version of the *a-priori* algorithm. If so, PAVE's bottleneck could be further reduced. It is also unclear if the *a-priori* step was a bottleneck in AuthorMiner's implementation. It is possible that the speed of our implementation allows the *a-priori* method to now be a bottleneck. Despite this, our total authorship process took about five minutes. Future work will concentrate on creating a parallel *a-priori* strategy to incorporate into PAVE.

We were further surprised to discover that when run on a random subset of the Enron database, we were unable to to reproduce AuthorMiner's results. What did the subset of e-mails used by AuthorMiner look like? When we queried the authors of AuthorMiner about this, we did not get a response. When we create a subset consisting of e-mails with a hundred words or more, we were able to finally achieve the accuracy numbers reported in the AuthorMiner paper. Further exploration is necessary to fully understand why this is the case. In the future, we also plan on exploring alternate approaches to gain high accuracy on e-mails with a smaller length.

Acknowledgments

We are sincerely grateful to the ARL Faculty and Cadet Collaborative Research Program and the DARPA Undergraduate Research Opportunity Program for their funding and support of this work. We would also like to thank the Mathematical Science Center and the Cyber Research Center, and the USMA High Performance Computing Center for their support and coordination of funding. Special thanks to LTC Craig Quadrato for his assistance in getting us access to DOD cluster resources, and to ERDC for generously granting us hours on the Garnet system. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the U.S. Military Academy or the U.S. Army.

References

[1] D. I. Holmes and R. S. Forsyth, "The federalist revisited: New directions in authorship attribution," *Literary and Linguistic Computing*, vol. 10, no. 2.

[2] E. Decarlo, "The view from here: Anonymity - a thing of the past?" *Math Horizons*, vol. 21, no. 4.

[3] R. S. Forsyth and D. I. Holmes, "Feature-finding for text classification," *Literary and Linguistic Computing*, vol. 11, no. 4, pp. 163–174, 1996.

[4] E. Stamatatos, "A survey of modern authorship attribution methods," *Journal of the American Society for information Science and Technology*, vol. 60, no. 3, pp. 538–556, 2009.

[5] J. Li, R. Zheng, and H. Chen, "From fingerprint to writeprint," *Communications of the ACM*, vol. 49, no. 4, pp. 76–82, 2006.

[6] A. Abbasi and H. Chen, "Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace," *ACM Transactions on Information Systems (TOIS)*, vol. 26, no. 2, p. 7, 2008.

[7] F. Iqbal, R. Hadjidj, B. C. Fung, and M. Debbabi, "A novel approach of mining write-prints for authorship attribution in e-mail forensics," *Digital Investigation*, vol. 5, pp. S42–S51, 2008.

[8] B. Klimt and Y. Yang, "The Enron corpus: A new dataset for email classification research," in *Machine learning: ECML 2004*. Springer, 2004, pp. 217–226.

[9] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[10] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[11] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating MapReduce for multi-core and multiprocessor systems,” in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*. IEEE, 2007, pp. 13–24.

[12] J. Talbot, R. M. Yoo, and C. Kozyrakis, “Phoenix++: modular MapReduce for shared-memory systems,” in *Proceedings of the second international workshop on MapReduce and its applications*. ACM, 2011, pp. 9–16.

[13] R. Zheng, J. Li, H. Chen, and Z. Huang, “A framework for authorship identification of online messages: Writing-style features and classification techniques,” *Journal of the American Society for Information Science and Technology*, vol. 57, no. 3, pp. 378–393, 2006.

[14] F. Bodon, “A fast apriori implementation,” in *Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations (FIMI03)*, 2003.