Oases: An Online Scalable Spam Detection System for Social Networks

Hailu Xu, Liting Hu, Pinchao Liu, Yao Xiao, Wentao Wang, Jai Dayal[‡], Qingyang Wang[†], Yuzhe Tang[§]
Florida International University,[‡] Intel Corporation, [†]Louisiana State University, [§]Syracuse University
Email:{hxu017, lhu, pliu002}@cs.fiu.edu, jai.dayal@intel.com qywang@csc.lsu.edu, ytang100@syr.edu

***Abstract*—Web-based social networks enable new community-based opportunities for participants to engage, share their thoughts, and interact with each other. Theses related activities such as searching and advertising are threatened by spammers, content polluters, and malware disseminators. We propose a scalable spam detection system, termed Oases, for uncovering social *spam* in social networks using an *online* and *scalable* approach. The novelty of our design lies in two key components: (1) a decentralized DHT-based tree overlay deployment for harvesting and uncovering deceptive spam from social communities; and (2) a progressive aggregation tree for aggregating the properties of these spam posts for creating new spam classifiers to actively filter out new spam. We design and implement the prototype of Oases and discuss the design considerations of the proposed approach. Our large-scale experiments using real-world Twitter data demonstrate scalability, attractive load-balancing, and graceful efficiency in online spam detection for social networks.**

***Keywords*-online social networks; spam detection; DHT;**

I. Introduction

The past few years have seen the rapid rise of Web-based systems incorporating social features, such as online social networks (OSNs) (e.g., Facebook, Twitter). These systems have a common feature that they rely on users as the primary source of posts and enable users to comment on others' posts. Unfortunately, such openness and reliance on users also attract *social spammers*, who advertise commercial spam messages, and disseminate malware [10]. Reports show that nearly 10% of tweets on Twitter are all spam [1], and Facebook usually blocks 200 million malicious actions every day [16].

We observed that social spammers who aim to advertise their products or post victim links are more frequently spreading malicious posts during a very short period of time. Fig. 1 shows the three days' social rumor activities that are extracted from a real-world dataset of the Charlie Hebdo shooting in 2015 [28]. X-axis and Y-axis present the time and the number of rumors, respectively. Each peak with a color presents the activities of one specific rumor. We can observe that: (1) once spam post is produced, it spreads in a very short period of time and will soon reach its peak; (2) the content of spam post is always "drifting", and multiple peaks in different colors indicate that the contents of social spam change rapidly.

Besides, recent surveys and research reported that social spam is normally fast changing, and spam activities are usually concentrated in a short period of time [2, 7, 26].

Therefore, *the major challenge for the spam detection system is enabling the update of trained classifiers to keep pace of the collection of spam information promptly, so as to uncover and defend against these social spammers.*

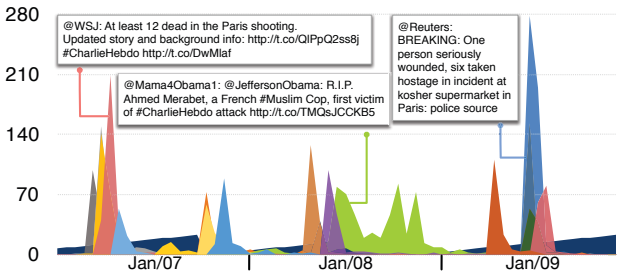


Figure 1. Social rumors of the famous Charlie Hebdo shooting in 2015, Twitter [28]. We present the activities of 14 rumors in the figure and show several example statements.

Traditional techniques for discovering evidence of spam and spammers have the following limitations: (1) they mainly focus on analyzing offline historical logs [11, 19, 20, 22, 24], limiting the capability to adapt to the new spam emergence and resulting in failing to uncover the most recent spam; and (2) they mainly focus on centralized data sources and ignore the fact that most OSN logs are continuously generated in distributed web servers [11, 12, 17, 22], limiting the capability to take the advantage of continuously processing the distributed data on the fly and resulting in centralized bottleneck and load unbalance.

From the data mining point of view, the spam detection has three major steps: (1) model construction where the spam classifier is created using a training dataset with a specific algorithm, e.g., Random Forest [4]; (2) model test where the test dataset is used to validate the accuracy of the spam classifier; and (3) use well-trained classifier with new social data to get predictions that identify new spam.

We propose a novel **O**nline sc**a**lable **s**pam d**e**tection **s**ystem, namely **Oases**. The key idea of **Oases** is to enhance the ***online*** feature and ***scalable*** feature into the general spam detection processing, in which the spam detection model is continuously constructed with the online training dataset and model testing, and the spam detection is performed in a scalable fashion.

Oases operates in two phases. The first phase is the spam model construction. We build a distributed hash table (DHT) [13] based aggregation tree to feed the distributed data sources into a decentralized peer-to-peer overlay, which consists of a root, branches and many leaves. The root is responsible for disseminating the continuously updated training dataset and test dataset to the branches and leaves. Each leaf node is responsible for spam model construction and spam model test by applying the training dataset to a specific classifier such as Random Forest algorithm [4], and testing the spam model using the test dataset. The intermediate

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00020

results are aggregated by branches to the root for validation and confirmation.

The second phase is the spam model application. Each leaf takes the new coming streaming events from the distributed data sources (e.g., Twitter logs, Facebook logs, etc.), analyzes them using the spam detection model from the first phase, outputs spam and labels. The spam with their labels are then aggregated to the root and reported to the end users.

The novelty of our work lies in that the posts are progressively aggregated for actively filtering out new spam and publishing the training dataset to all distributed leaf agents to update the classifiers in an online and scalable fashion. We believe our system could promptly detect activities of spammers and classify various latest spam for social networks.

This paper makes the following technique contributions:

- An ***online*** spam detection system called **Oases** is presented that defend against new changes of spam actions.
- A ***scalable*** DHT-based overlay with spam detection is presented. It orchestrates the independent processing of geographically distributed agents and aggregates their intermediate results into the final results.
- A comprehensive evaluation of Oases performance and functionality on a large cluster using real-world Twitter data is presented.

Experimental results show that Oases achieves graceful performances in scalability and the online spam detection accuracy. And it achieves attractive load balancing in scaling with hundreds of agents and millions of social posts. Data classification also shows good performances with the F1 score up to 96% and accuracy up to 94%. Besides, the system presents consisting performances in the runtime overheads and resource consumption when scaling from dozens to hundreds agents.

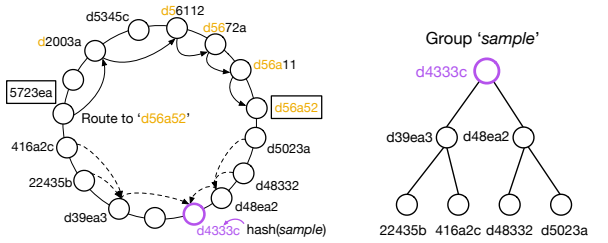
The rest of this paper is organized as follows: Section II describes the background of our work. Section III presents the design and functional components of Oases. The evaluation results using real-world logs are shown in Section IV. Section V presents the related works. Finally, we describe the conclusions from this work in Section VI.

II. Background

Inspired by the success of DHT-based decentralized overlay in peer-to-peer networks [6, 13], Oases uses DHT-based routing protocols to (1) build a DHT-based aggregation tree to feed the distributed data sources into a decentralized peer-to-peer overlay for harvesting and uncovering spam and (2) rapidly inform distributed agents about the updated classifiers. In this section, we briefly discuss the background of our work.

A. Pastry Decentralized Overlay

In Pastry [13], each participating node is assigned with an identifier (nodeId) that is used to identify node and route message. Given a message and a key, the message can be guaranteed to be routed to the node with the nodeId numerically closest to that key, within $\lceil log_{2^b} N \rceil$ steps (default $b = 4$). Each node maintains a routing table and leaf set



(a) Details of message route. (b) The tree of a sample group.

Figure 2. (a) Routing a message from node *5723ea* to node *d56a52*. (b) Node *d4333c* serves as the rendezvous node of the group '*sample*', with nodeId numerically closest to the groupId.

to support the message routing, self-organization, and fault recovery functionalities.

1) Message routing: As shown in Fig. 2a, at each routing step, a node typically forwards a message to a node whose nodeId shares with the key a prefix of at least one (or more) digit longer than the given nodeId. If no such a node, it forwards the message to the node which nodeId shares the prefix with the key same as the current node, but is numerically closer to the key.

2) Routing table and leaf set: The routing table consists of node prefixes (IP address, latency information, and Pastry nodeId) arranged in rows by the common prefix length. The leaf set for a node contains a fixed number of nodes that have the numerically closest nodeIds to that node. This assists nodes in the last step of routing messages and in rebuilding routing tables when nodes fail.

B. Scribe's Group Management

Scribe is an application-level group communication system built upon Pastry. Scribe manages groups and maintains a spanning tree containing the members of each group. Fig. 2b presents a tree of group "*sample*". All nodes join Pastry, and subsequently, nodes may join and leave groups. Scribe can handle group sizes varying from one to millions, and it efficiently supports rapid changes in group membership [6, 13]. It uses a pseudorandom Pastry key to name a group, called groupId. Usually, the groudId is the hash of the group's textual name concatenated with its creator's name. To create a group, a Scribe node asks Pastry to route a CREATE message using the groupId as the key. The node responsible for that key becomes the root of the group's tree. To join a group, a node routes a JOIN message towards the groupId. The message will continue to be routed till it reaches a node in that tree. The route traversed by the message to the group multicast tree would be added. As a result, Scribe can efficiently support large numbers groups, arbitrary numbers of group members, and groups with highly dynamic membership.

Scribe supports two major properties: multicast and any-cast. We use multicast to develop a hierarchical aggregation tree: a fundamental abstraction for scalability of Oases. Multicast sends the message to every group members and messages are disseminated from the rendezvous point along the group tree. Any-cast is implemented using a distributed depth-first search of the tree. Any node in the overlay can any-cast to a Scribe group by routing the message towards the groupId.

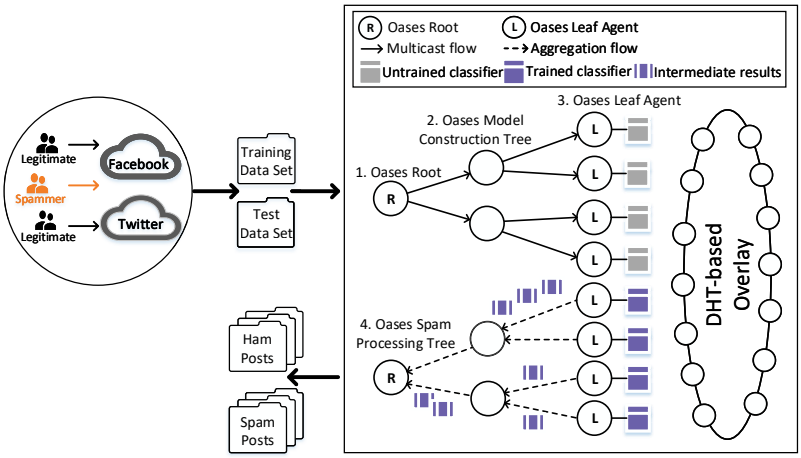


Figure 3. Overall framework of Oases. Oases consists of four components: the Oases root, the leaf agent, the Oases model construction tree, and the Oases spam processing tree. The Oases root first manually separates the collected raw data into training and test datasets. Second, the Oases root disseminates those datasets to the leaf agents via the Oases model construction tree. Third, Oases leaf agents complete the data training processing. Besides, they connect to the web servers to gather online social activity logs. Fourth, the Oases spam processing tree accomplishes the spam detection. By organizing the leaf agents to complete spam mining tasks in a scalable manner, detected *Spam* and *Ham* (non-spam) posts are rolled up level by level via the tree until reach the root. Finally, the classified results are transmitted to the users.

III. Oases Design

In this section, we introduce the Oases system, discuss each functional component of the system, and outline the details of workflows in the Oases system.

A. Oases Overview

As shown in Fig. 3, the Oases system consists of four major components: (1) the Oases root; (2) the Oases model construction tree; (3) the Oases leaf agent; and (4) the Oases spam processing tree.

The first component is the Oases root. The Oases root is responsible for the main control flows on other nodes, e.g., publishing the instructions from the Oases root to branches and leaf agents to start training the classifier, delivering messages to the Oases leaf agents to start classifications, etc. As shown in Fig. 3, for the first step, the Oases root manually divides the raw dataset into training dataset and test dataset. The training dataset and test dataset are then disseminated by the Oases root to all distributed Oases leaf agents through the Oases model construction tree which is discussed next.

The second component is the Oases model construction tree. As shown in Fig. 3, for the second step, the Oases model construction tree is responsible for creating ***efficient paths*** for the Oases root to disseminate the training dataset and test dataset to the Oases leaf agents. The key idea is the use of a DHT-based application-level multicast tree [6] to disseminate copies in a progressively way following the tree path, without maintaining N point-to-point connections for N leaf agents.

The third component is the Oases leaf agent. The Oases leaf agent is responsible for the training of the spam detection model. As shown in Fig. 3, for the third step, the leaf agent applies the received training dataset to the Random Forest algorithm [4] to practice the classifier, and uses the test dataset to enforce the classifier. The trained classifier is used later by the fourth component to do the online spam detection.

The fourth component is the Oases spam processing tree. The Oases spam processing tree is responsible for orchestrating the distributed Oases agents to fulfill the data mining tasks of online spam detection in a scalable fashion. The Oases leaf agents are directly connecting to the web servers that generate user activity logs, i.e., tweets, and classify spams out of these logs. As shown in Fig. 3, for the fourth step, the workflow of Oases spam processing tree is as follows: a scalable aggregation tree "rolls up" the classified results from the Oases leaf agents level by level until the results reach the root. For example, if one tree has 7 spam processing agents and each leaf agent classifies 10,000 social data, then after aggregation, the root agent receives 70,000 classified results.

B. Oases Root

The Oases root is responsible for the main control flows of the whole system, including (1) dividing the raw dataset into training dataset and test dataset; (2) publishing the datasets from the Oases root to branches and leaf agents to start training the classifier; and (3) aggregating the spam detection intermediate results from the Oases leaf agents to the root.

The Oases root uses a DHT-based hierarchical tree as the main channel for disseminating datasets and instructions. The DHT-based hierarchical trees are built as follows:

1) Step 1: constructing a peer-to-peer overlay leveraging Pastry [13]. Each Oases node is assigned a unique, 128-bit nodeId in a circular nodeId space ranging from $0 \sim 2^{128} - 1$. All nodes' nodeIds are uniformly distributed. Given a message and a key, the message can be guaranteed to be routed to the node with the nodeId numerically closest to that key, within $\lceil log_{2^b} N \rceil$ steps, where b is a base with a normal value 4.
2) Step 2: building a multi-cast tree leveraging Scribe [6] (more details can be found in Section III-E). Any node in the overlay can create a group with a groupId which is the *hash* (SHA-1) the group's name concatenated with its creator's name. Other nodes can join the group by routing a JOIN message towards the groupId. The

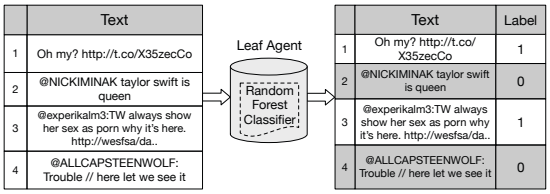


Figure 4. Sample of the data classification in the Oases leaf agent. Original test data has no labels. After classification, the predicted labels are generated. Label 1 presents *Spam* and 0 presents *Ham* (non-spam).

node which its nodeId is most near to the groupId serves as the root. The tree multicasts a message to all members of the group within $O(logN)$ hops.

3) Step 3: enhancing the aggregation function on branches. The Oases root and middle-level nodes jointly implement (1) the aggregation flow and (2) the control flow. For example, for the social spam detection application, batches of social logs are parsed as a map from hashed data contents (*ID*) to the classified tags (*labels*), i.e., (*DDA2*, 1) and (*F7B5*, 0) in the Oases leaf agent (here the classified tag 1 means the data is classified as spam, tag 0 represents non-spam, and we use shortened hashes to indicate *ID*). Then the aggregation tree that progressively 'rolls up' and reduces those *ID-label* pairs from the distributed leaf agents to the root (more details can be found in Section III-F). Besides, when necessary, the Oases root can multicast to its workers within the group, to notify them to empty their sliding windows and/or synchronously start a new batch.

C. Oases Leaf Agent

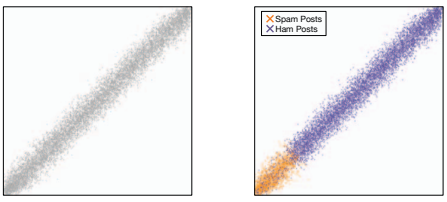
The Oases leaf agent is responsible for the local data processing task by executing the root's instructions. Data processing task in the leaf agent consists of two roles: (1) local data classification and; (2) local online social spam detection.

The local data classification is the first role of the leaf agent. Each leaf agent trains a classifier by using the training dataset. Then it uses the test dataset to examine the accuracy of the trained model. Besides, the leaf agent updates its trained model periodically with the new delivered training and test datasets from the root. This allows the trained model to detect spam efficiently with the latest spam features.

Fig. 4 shows the processing of data classification in a leaf agent. Original social data is normalized in dataset without labels. After the classification via the trained model, each instance of the original data acquires a classified label, which identifies spam or not.

Fig. 5 shows the visualization of data classification in one leaf agent. Fig. 5a shows the original dataset without predicted labels. After the classification, this dataset is classified as two groups, as shown in Fig. 5b, where the purple and orange color represent *Ham* (non-spam) and *Spam*, respectively.

The local online social spam detection is the second role of the leaf agent. In Oases, each leaf agent connects to a web server so as to collect the online social streaming data from this server. Then the leaf agent completes the online data analysis upon streaming data flow with the trained model and produces classified results. Finally, all leaf agents collaborate



(a) Unclassified data. (b) Classified data with labels

Figure 5. Visualizaiton of the data classification in the Oases leaf agent. (a) shows the original dataset which has no labels. (b) shows the dataset with labels predicted by the model.

to shuffle the classified results to the upper layer via the spam processing tree. More details can be found in Section III-F.

D. Oases Classified Algorithm

We next introduce the details of the classic algorithm that be implemented, Random Forest algorithm [4], in our training and test processing.

Why Random Forest? Random Forest algorithm is a classic data mining algorithm and had been implemented with graceful performances in various works of social spam detection [21, 24]. Random Forest constructs a fixed number of decision trees for training during the training processing and results in one final decision which is determined from multiple individual trees. This algorithm is derived from decision tree learning and tree bagging.

In the training process, the classifier in each leaf agent receives the training dataset from the root agent, then randomly samples *N* cases to create a subset of the data. The subset usually about 66% of the total set. One subset of the samples creates one decision tree. That is repeatedly to choose some different small subset of attributes at random and creates all decision trees. When leaf agent starts the test processing, trained classifier puts the test dataset into the forest. Then it runs down all trees of the forest. The classification result is the majority vote among all decision trees.

E. Oases Model Construction Tree

The Oases model construction tree is responsible for creating ***efficient paths*** for the Oases root to disseminate the training and test dataset to the Oases leaf agents. Here we use an example to illustrate the Oases model construction tree. The sample scenario is presented in Fig. 6. Assume there are 7 nodes in the Oases system. The node with nodeId numerically closest to the *topicId* acts as the rendezvous point for the associated multicast tree. For example, if *hash*(model) equals to *0088*, the node with same identifier or closest identifier like *0087* or *0089* will be the root of the model tree. In Fig. 6, the tree is rooted at the rendezvous point and the other nodes subscribe to this tree. The Oases root multicasts the training and test datasets to all leaf agents in $O(logN)$ hops. Then those leaf agents are triggered to apply the received dataset to the local classifier to complete the model training and test processing using the Random Forest algorithm [4].

F. Oases Spam Processing Tree

The Oases spam processing tree is responsible for coordinating distributed leaf agents to accomplish the online spam

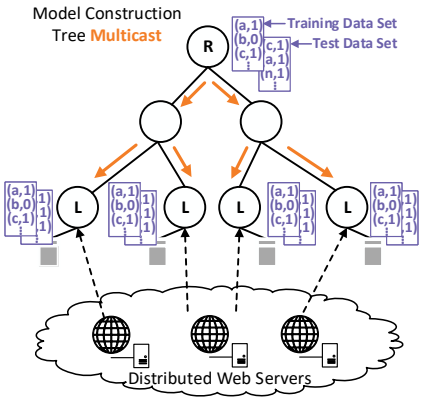


Figure 6. Oases Model Construction Tree. In this tree, the Oases root multicasts the training and test datasets to the leaf agents via the efficient paths that the tree offered within $O(logN)$ hops.

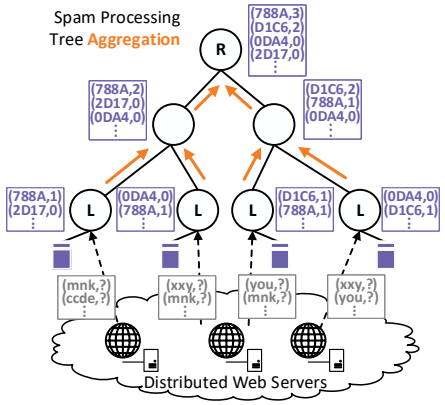


Figure 7. Oases spam processing tree. Each leaf agent collects social streaming data from web servers. Then its trained model processes data and produces the classified results, and sends the results to the upper layer.

detection globally. In this section, we use a sample scenario to present the workflow of the Oases spam processing tree.

As shown in Fig. 7, the leaf agent processes the online social streaming data with its trained classifier. For instance, after processing, original data (*mnk*, ?), in which the question mark means it hasn't been classified, is detected as spam and marked as (*mnk*, 1). Then the leaf agent sends the hashed content, e.g., *hash*(*mnk*) = *788A* and its label formatting as (*788A*, 1), to the upper layer.

Further, the spam processing tree progressively rolls up and reduces those ID-label pairs from the distributed leaf agents to the root. For example, ⟨(*788A*, 1), (*2D17*, 0)...⟩, ⟨(*0DA4*, 0), (*788A*, 1)⟩, are reduced as ⟨(*788A*, 2), (*2D17*, 0), (*0DA4*, 0)...⟩ in the branches of tree. And then those pairs are reduced as ⟨(*788A*, 3), (*D1C6*, 2), (*2D17*, 0), (*0DA4*, 0)...⟩ to the root as the final results. The value of labels indicates the number of leaf agents which detect this data as spam, e.g., (*788A*, 3) represents that there are 3 leaf agents classifying the data "*mnk*" as a spam post. Those hashed IDs with larger values in labels indicate the higher possibility as spam posts.

G. Self-adjustable Tree

Oases supports self-tuning in the tree structure level. By manipulating the parameter *n* of the tree fan-out, with achieving 2^n fan-outs per agent, it can format different trees.

The design of this feature is to support multiple targets in spam processing. For example, when an application is high latency sensitivity, it can modify the tree depth by adjusting the value of tree fan-out. Assuming there are 10^b agents in Oases, the default depth of the tree is $log_{2^b}N$, where b = 4. By changing the default fan-out from 2^4 to 2^5, the average depth of the tree is reduced from 5 to 4. So root-to-leaf data transfer can achieve lower latency by across fewer layers. When an application desires a good failure recovery, Oases can increase the depth of trees by reducing the tree fan-out. Using the same example above, Oases can change the fan-out from 32 to 16, resulting in that a tree's depth increases from 4 to 5. A deeper tree can benefit the agent's failure recovery. This depends on the mechanism for failure recovery in Oases: once a child fails to receive a heartbeat message, it suspects its parent failed, and this agent will route the JOIN message to the group's identifier. Oases then sends the message to a new parent to repair the tree. When a tree has a small fan-out and a large depth, the failure of one agent can only affect the performances of the following sub-agents, while fewer sub-agents can reduce this failure effects.

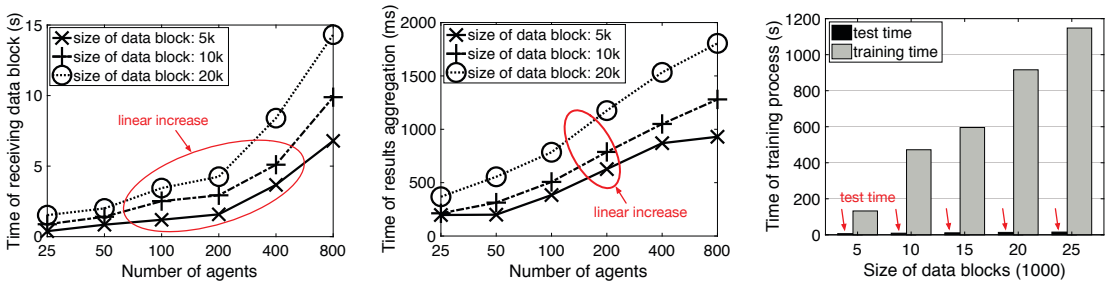
H. Benefits and Design Rationale

In this section, we discuss why Oases has the online and scalability benefits and the rationale behind the design.

Online. Oases enables the progressive aggregation of the properties of the spam posts for creating new spam classifiers to actively filter out new spam posts and update the classifiers to all distributed data process agents. That ensures the spam classifiers to always keep pace with the latest social spam, and identify new spam with high efficiency.

Exploring DHTs for Scalability. The Oases model construction tree and spam processing tree are self-organizing and self-repairing, and can be easily expanded in a distributed manner. The use of DHT guarantees that the cost of multicast and aggregation can be fulfilled within $O(logN)$ hops. Moreover, multiple groups are supported in one single overlay, which means that the overhead of maintaining a complex overlay can be amortized over all groups' spinning trees [13]. Specifically, all agents in overlay are viewed as equal peers, so, each agent can be a root, parent, leaf agent or any combination of the above, which leads to well balance of the computation loads.

Handling Nodes' Failures. The Oases system uses leaf sets to handle node failure [6]. Each node maintains a leaf set. The leaf set is the set of *l* nodes which nodeIds that are numerically closest to the present nodeId, with *l*/2 larger and *l*/2 smaller. A typical value of *l* is nearly $8\lceil log_{2^b}N \rceil$, where N is the total number of nodes in the system. Neighboring nodes in node's leaf set exchange keep-alive messages periodically. An agent is presumed as a failure if it is unresponsive for a period. Then those members in the left set of failed node's leaf set are notified and they update their leaf sets. Once the node recovers, it will contact the node in its last known leaf set, obtain their current leaf sets, update its own leaf set and then notify the members in the new leaf set of its recovery.



(a) Receive time of data blocks. (b) Delivery time of results. (c) Time of training and test process.

Figure 8. Performance evaluation of Oases on time of data delivery, receive, and training test process. (a) shows the time of the leaf agents receiving data blocks from root agent with various size of data blocks and different number of agents. (b) represents the time of shuffling data from leaf agents to upper layer in the tree. (c) shows the training and test time in data processing with different size of data blocks.

IV. EVALUATION

We evaluate the Oases system with the real-world online social network streaming data. Experimental evaluations answer the following questions:

- What are the spam detection accuracy rates of the Oases system (Sec. IV-B)?
- What are the performances of data shuffling, processing and delivery latency in Oases (Sec. IV-C & Sec. IV-D)?
- What is the overhead and resource consumption of the system at runtime (Sec. IV-E)?

A. Testbed and Application Scenarios

Experiments are conducted on a testbed of 800 agents hosted by 16 servers running on Linux. Each server has a QEMU Virtual CPU with 3.4GHz processor, 4G of memory and 30 GB hard drives. The system was implemented in Java by using Java SE Development Kit 7 in x64, version 1.7.

Oases's functionality is evaluated by running an online social data application. Nearly 3,000,000 tweets from Twitter streaming API had been collected and evaluated via our system from 12.2016 to 02.2017. We use the straightforward content features (URL, words, etc.) to predict labels. Oases uses test dataset to examine the model which is trained with training dataset. The application is implemented to predict labels from online data streams via the Oases leaf agents.

B. Spam Classification Results

We evaluate the spam classification results of Oases and compare the performance with several popular classifiers. Evaluations rely on a sample dataset which consists of 50,000 posts (37465 posts are *Ham* and 12535 posts are *Spam*). The results are shown in Table I. Random Forest is default implemented in Oases and other classifiers include K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Logistic Regression, and Naive Bayes. F1-score (F-Measure) responses for an important factor in measuring the classification performance. The results show that Random Forest achieves promising performance with the F-measure up to 96.2% and the accuracy up to 94.8%. Combined with these key indicators, Random Forest achieves the best performance among all classifiers.

Table I. RESULTS OF SPAM CLASSIFICATION WITH MULTIPLE CLASSIFIERS.

Classifiers	Accuracy	F1	FPR
Random Forest	**94.8%**	**0.962**	**0.26**
SVM	94.5%	0.937	0.446
KNN	91%	0.911	0.374
Logistic	92%	0.908	0.303
Naive Bayes	86%	0.871	0.477

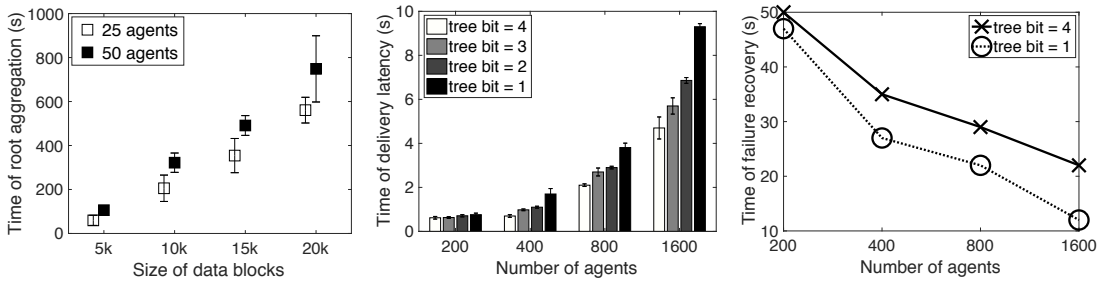
C. Data Shuffling Time and Data Processing Time

The Oases model construction tree, as the efficient paths for dataset distribution, directly influences the local data processing time. Besides, after the data processing, the classified intermediate results propagate from distributed leaf agents, to the upper layer for aggregation, until they reach the Oases root. The aggregation tree, as the structure for shuffling results from the leaves to the root, directly influences total spam processing latency. Therefore, we first report the time of the leaf agents receiving data blocks (datasets) from the root in Fig. 8a, and then report the time of intermediate results aggregating from the leaves to the root in Fig. 8b. Finally, we show the data processing time of each leaf agent in Fig. 8c.

Data Shuffling Time. Here we classify the data shuffling time into two parts: (1) the time of the leaf agents receiving data blocks from the root; and (2) the time of results aggregating from the leaf agents to the root. The number of the Oases agents varies from 25 to 800. Simultaneously, various sizes of data blocks are used for evaluation.

Fig. 8a and Fig. 8b show that, when the system uses the same datasets but with a different number of agents, the time of delivery and reception linearly increases, rather than fold increases. This is because that the linear increment of the delivery or reception time is strictly determined by the tree depth $O(logN)$, which further reflects that the tree topology in the overall performance exhibits a very good balance.

Data Processing Time. Fig. 8c shows the time of model training and test processing in one agent with various sizes of data blocks. Result shows that with the increment of the size of data blocks, the training processing time also increasing rapidly, especially when the data block has 20*k* and 25*k* posts. It indicates that over-large size of data blocks can be the bottleneck of the whole system when considering the size reaches to 25*k* with the training time up to 1150s. Costly time in the training processing will cause the whole system looks like in "busy-waiting" - though the leaf agent is working on the training processing, the root cannot get any useful results in a long time. Therefore, the choice of a suitable size of the data block can promote the best performance of the system.



(a) Latency of results aggregation. (b) Average delivery latency of data. (c) Average fault recovery latency.

Figure 9. Latency of root aggregation, data delivery and fault recovery latency with different tree structures. (a) shows the average latency of root agent aggregates whole results in one process cycle. (b) shows the average data delivery latency across the different trees. The *tree bit* decides the fan-out of each agent with 2^n. (c) represents the average fault recovery latency of failed agents in different trees.

D. Aggregation Latency and Self-adjustable Tree

Performance impact due to large data blocks. After the hashed ID-label pairs are shuffled to the upper level, the Oases root actively aggregates the data stream into the final result pool. Although the Oases architecture ensures that the aggregation processing can be completed with $logN$ hops. However, many factors may impact the performance of the root aggregation, such the size of the data blocks, the network bandwidth, and the traffic interference.

As shown in Fig. 9a, the size of the data blocks (training and test datasets) has an important effect on the latency. In the case of reasonable dataset size, e.g., 5*k*, the average latency is nearly 100 seconds. However, when using large data blocks (e.g., 20*k*), the latency grows much faster than the size increment of the data block.

We believe that the increased latency indicates that the system has reached a limited overload when processing with extra large data blocks. In this case, some agents are still active but other agents may be blocked to wait for the server's resources. In addition, oversized dataset exacerbates the burden of each leaf agent during the training and testing processing that causes the overload even further overload.

Tree Structure Adjustment. Fig. 9b and Fig. 9c represent the performances of delivery latency and recovery latency with different tree structures. In Fig. 9b, *tree bit* decides the tree fan-out of each agent. For example, when *tree bit* = 4, the fan-out is 16, which means each agent has 16 following agents. Results in Fig. 9b show that delivery latency increases when the tree layer increases (small *tree bit*), in which delivering a data block from the leaf to the root need to cross more layers.

Fig. 9c presents the relationship between the failure recovery and the tree structure. Results show that when the tree layer is small (large fan-out), the latency of agent failure recovery increases. That is reasonable as we present before. When the tree fan-out gets larger, once an agent fails, more child agents need to recover their functions and structures, which will take a longer time to process and find a new parent. Besides, with the increment of tree agents, the average latency decreases. This proves that when the tree becomes deeper, the locality properties of Pastry lead to shorter routes and benefit the recovery of failed agents [13].

Table II. THE RUNTIME OVERHEADS OF OASES. IT REPRESENTS THE CPU, MEMORY, I/O, AND CONTEXT SWITCH OVERHEADS.

VSD	CPU	Memory	I/O	C-switch
	%used	*%used*	*wtps*	*cswsh/s*
5k	47.6%	38.0%	2.23	322.72
10k	51.1%	43.8%	3.20	280.69
15k	51.0%	44.2%	2.78	304.57
20k	50.9%	45.1%	1.80	314.06

VSD: various size of data blocks.
wtps: write transactions per second.
cswsh/s: context switches per second.

E. Runtime Overhead

Table. II shows the runtime overhead of Oases. As the result shows, the Oases system has similar overheads in the utilization of CPU, memory, I/O, and context switches when dealing with the data blocks of different sizes. This is because the Oases system uses a decentralized architecture to distribute the management load evenly over the distributed servers, and the hierarchical tree structure facilitates communication across multiple agents and servers.

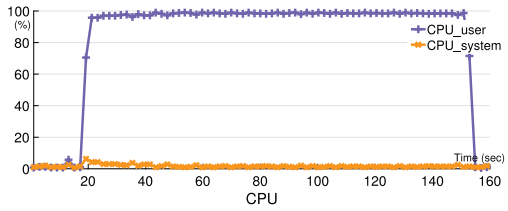
We also evaluate the server's resource consumption, with each server supporting five leaf agents. As shown in Fig. 10, the processing cycle time is close to 140 seconds, with the CPU and memory utilization reaching a higher level from 15s to 155s. In addition, the processing performance is quite consistent with the former results in Fig. 8.

V. RELATED WORK

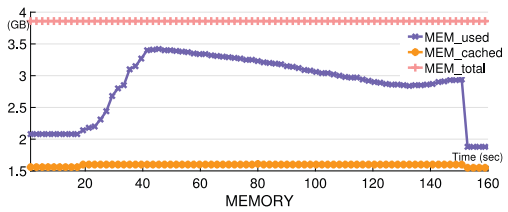
A. Spam Detection in Social Networks

Offline. Many former studies had focused on offline methods in analyzing social data [11, 15], which utilized a limited historical dataset. Based on the connection between users and user trusts, [17] identifies victims in Twitter with a dataset that was collected in 10 months. By analyzing several historical datasets, Spade [22] presents that new spam can be detected from one social network across other social networks. Prior studies classified social spam from various perspectives, either from the view of polluted contents [11], user behaviors [27], or from hashtags [15], inherent features [21]. However, these studies still limit in a specific size of historical data and are difficult to catch up the online latest features of social spam.

Centralized Processing. Former studies normally focused on centralized processing [12, 17]. [8] presents an online spam filtering framework in a central server by using spam



(a) CPU utilization in one server



(b) Memory utilization in one server.

Figure 10. CPU and memory utilization in one server.

campaigns. TopicSketch [23] is a real-time framework that combines a sketch-based topic model and a hashing-based dimension reduction to detect bursty topics. Lfun [7] is a real-time statistic features-based system which can extract spam from social drifting data. Monarch [18] utilizes the online URL blacklists to detect URL spam in real-time. The difference between our approach and these studies is that they normally focused on centralized spam analysis, while we focus on social spam detection in distributed manner.

B. Distributed Social Data Processing

Recent applications had been cooperated with scalable methods to achieve efficient processing [3, 14]. [5] presents a parallel spam filtering system based upon MapReduce. To mitigate the accuracy degradation by parallel SVM, they augment with ontology semantics. Different with them, we allow data training can be completely implemented in local agents and maintain the desired accuracy. ELF [9] is a decentralized model for the streaming process and supports powerful programming abstraction for batch, iterative and streaming processing. SSTD [25] is a dynamic truth discovery scheme which can discover Twitter data truth with scalability. Different from above work, we use the latest data features as the feedback and analyze social spam in a scalable way.

VI. Conclusion

In this paper, we present the online scalable spam detection system (Oases), a distributed and scalable system which detecting the social network spam in an online fashion. By periodically updating the trained classifier through a decentralized DHT-based tree overlay, Oases can effectively harvest and uncover deceptive online spam posts from social communities. Besides, Oases actively filters out new spam and updates the classifiers to all distributed leaf agents in a scalable way. Our large-scale experiments using real-world Twitter data demonstrate scalability, attractive load-balancing, and graceful efficiency in online spam detection. Future work on Oases will go beyond additional implementation steps, e.g., to implement new specification/configuration APIs for end users, achieve high-availability by exploring checkpointing/failover approaches, reduce runtime overhead, all with goals of achieving both good performance and high resource efficiency for large-scale online spam detection.

Acknowledgment

We gratefully thank the anonymous reviewers for their feedback and thank Florida International University School of Computing & Information Science and GPSC for the support to present this work.

References

[1] "Almost 10% Of Twitter Is Spam". https://www.fastcompany.com/3044485/almost-10-of-twitter-is-spam%20from%20your%20cite.
[2] "Avoiding Social Spam Hackers on Facebook and Twitter". https://www.socialmediatoday.com/news/social-media-today-predictions-for-2018-infographic/513179/. 2018.
[3] Janki Bhimani et al. "Performance prediction techniques for scalable large data processing in distributed MPI systems". In: *IPCCC, 2016*.
[4] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001).
[5] Godwin Caruana, Maozhen Li, and Man Qi. "A MapReduce based parallel SVM for large scale spam filtering". In: *FSKD, 2011*.
[6] Miguel Castro et al. "SCRIBE: A large-scale and decentralized application-level multicast infrastructure". In: *IEEE JSAC* (2002).
[7] Chao Chen et al. "Statistical features-based real-time detection of drifted Twitter spam". In: *IEEE Trans. Inf. Forensic Secur.* (2017).
[8] Hongyu Gao et al. "Towards Online Spam Filtering in Social Networks." In: *NDSS*. Vol. 12. 2012, pp. 1–16.
[9] Liting Hu et al. "ELF: Efficient Lightweight Fast Stream Processing at Scale." In: *USENIX Annual Technical Conference*. 2014, pp. 25–36.
[10] Kyumin Lee, James Caverlee, and Steve Webb. "Uncovering social spammers: social honeypots+ machine learning". In: *SIGIR*. 2010.
[11] Kyumin Lee, Brian David Eoff, and James Caverlee. "Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter." In: *ICWSM*. 2011.
[12] Michael Mathioudakis and Nick Koudas. "Twittermonitor: trend detection over the twitter stream". In: *ACM SIGMOD, 2010*.
[13] A. Rowstron and P. Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems". In: *ACM Middleware 2001*.
[14] Shweta Salaria et al. "Evaluation of HPC-Big Data Applications Using Cloud Platforms". In: *IEEE/ACM CCGrid, 2017*.
[15] Surendra Sedhai and Aixin Sun. "Effect of Spam on Hashtag Recommendation for Tweets". In: *WWW, 2016*.
[16] "Social Media Today Predictions for 2018". http://www.sileo.com/social-spam/. 2015.
[17] Kurt Thomas et al. "Consequences of connectivity: Characterizing account hijacking on twitter". In: *ACM CCS, 2014*.
[18] Kurt Thomas et al. "Design and evaluation of a real-time url spam filtering service". In: *Security and Privacy (SP), IEEE, 2011*.
[19] Courtland VanDam and Pang-Ning Tan. "Detecting hashtag hijacking from twitter". In: *ACM Web Science*. 2016.
[20] Bimal Viswanath et al. "Towards Detecting Anomalous User Behavior in Online Social Networks." In: *USENIX Security Symposium*. 2014.
[21] Bo Wang et al. "Making the most of tweet-inherent features for social spam detection on twitter". In: *arXiv:1503.07405* (2015).
[22] D. Wang, D. Irani, and C. Pu. "Spade: a social-spam analytics and detection framework". In: *Social Network Analysis and Mining* (2014).
[23] Wei Xie et al. "Topicsketch: Real-time bursty topic detection from twitter". In: *IEEE TKDE* (2016).
[24] Hailu Xu, Weiqing Sun, and Ahmad Javaid. "Efficient spam detection across online social networks". In: *ICBDA, 2016*.
[25] Daniel Yue Zhang et al. "Towards scalable and dynamic social sensing using a distributed computing framework". In: *ICDCS*. 2017.
[26] Z. Zhao, P. Resnick, and Q. Mei. "Enquiring minds: Early detection of rumors in social media from enquiry posts". In: *WWW*. 2015.
[27] Rongda Zhu et al. "Exploiting temporal divergence of topic distributions for event detection". In: *IEEE Big Data*. 2016.
[28] Arkaitz Zubiaga et al. "Analysing how people orient to and spread rumours in social media by looking at conversational threads". In: *PloS one* 11.3 (2016).