IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 December 2022, accepted 5 January 2023, date of publication 12 January 2023, date of current version 25 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3236604

RESEARCH ARTICLE

Achieving Online and Scalable Information Integrity by Harnessing Social Spam Correlations

HAILU XU[1], PINCHAO LIU[2,3], BOYUAN GUAN[2], QINGYANG WANG[4], (Member, IEEE), DILMA DA SILVA[5], (Senior Member, IEEE), AND LITING HU[6], (Member, IEEE)

[1]Department of Computer Engineering and Computer Science, California State University, Long Beach, CA 90840, USA
[2]School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA
[3]Meta Platforms, Inc., Menlo Park, CA 94025, USA
[4]Department of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA 70803, USA
[5]Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA
[6]Department of Computer Science and Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064, USA

Corresponding author: Hailu Xu (hailu.xu@csulb.edu)

This work was supported by the National Science Foundation (NSF) under Grant NSF-CAREER-2205677, Grant NSF-SPX-2202859, Grant NSF-SPX-1919181, Grant NSF-OAC-2212256, and Grant NSF-HDR-1934904.

ABSTRACT Malicious web links, social rumors, fraudulent advertisements, faked comments, and biased propaganda are overwhelmingly influencing online social networks. Enabling information integrity is a hot topic in both academia and industry. Traditional social spam detection techniques rely on centralized processing, focusing only on one specific set of data sources, thereby ignoring the social spam correlations between distributed data sources. In this paper, we propose an online and scalable misinformation detection system, named Spiral, to uncover social spam by leveraging the correlations between different social data sources in geo-distributed sites. The key insight in our approach is to amplify the effectiveness of state-of-the-art techniques to detect inappropriate posts by enabling the efficient large-scale propagation of detection information across domains. The novelty of our design lies in three key components: (1) a decentralized distributed hash-table-based tree overlay deployment for harvesting and uncovering deceptive information spreading in multiple online social networks communities; (2) a progressive aggregation tree for collecting the properties of these posts and creating new classifiers to actively filter out the propagation of inappropriate posts; and (3) a group communication structure that allows multiple groups to exchange the correlations among distributed social data sources. We designed and implemented a prototype of the Spiral system. Our large-scale experiments, using real-world social data, demonstrate Spiral's scalability, effective load-balancing, and efficiency in online spam detection for social networks.

INDEX TERMS Online social networks, information integrity, spam detection, distributed hash tables, stream processing.

I. INTRODUCTION

Online social networks (OSNs) have become indispensable for many people. More and more people receive the latest news, advertisements, social interactions, and burst topics from current popular OSNs such as Instagram, Facebook, Twitter, and WeChat. OSNs allow users to repost or comment on others' contents, which results in the prosperity of the virtual social world. Unfortunately, such openness and flexibility have turned the OSNs into serious threats for spreading deceptive news [74], fraudulent advertising [72], disseminating malicious software, phishing attacks, etc. For example, Twitter has found around 10 million misinformation post and manually removed around 5,000 accounts those who spread unreliable information about COVID-19 from Jan 2022 [27]. Until 2021, Facebook had removed 20 million posts that break COVID-19 misinformation policies and affected more than 2 billion users [19].

Social spam posts typically spread within a very short period. Figure 1 shows the diffusion for 20 minutes of the social fake news about COVID-19 vaccines [12]. Nodes with different colors indicate different fake news topics. Within

The associate editor coordinating the review of this manuscript and approving it for publication was Arianna Dulizia.

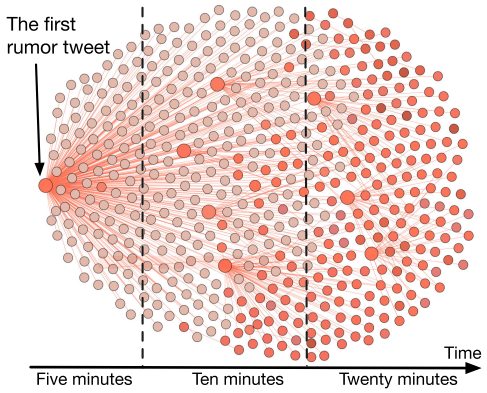


FIGURE 1. Spreading pattern for twenty minutes of spam related to the COVID-19 vaccines in Aug 2021. (Data source: [33]). Each node represents one account and two accounts are connected if one re-posts the other's fake news post.

20 minutes of the first fake news posting, there were charged comments and retweets propagating the posting. The dataset depicted in Figure 1 reveals that: (1) **spam posts have explosive propagation:** they blast within a short time and soon reach their peak rate of comments and retweets; (2) **"drifting" contents:** the content is always "drifting" with different faked details that relate to the same social event; (3) **spam correlations:** the social spam data produced from different social communities exhibits strong correlations in the contents, topics, and activities within the same (or closeby) time period, especially when a critical event or breaking news overwhelms the online social networks.

To keep pace with all spammers' activities, we need mechanisms to quickly update spam detection models in all social media sites (which may consist of hundreds of thousands of web servers in total). The key challenge is: *how can we uncover and defend against online spam activities, in real-time and at scale, by leveraging spam correlations between different social data sources in geo-distributed sites?*

Existing efforts have the following three limitations. First, they mainly follow the classic big data processing pipeline [32], [32], [67], [85], [87]: all social media streams are first collected by distributed log collection systems, transporting massive quantities of users' microblog stream data from geographically distributed web servers to a centralized storage tier. Only after all stream data has arrived do the data processing engines (e.g., Spark Streaming with MLlib [11]) start processing these data. Such a long path hugely delays the processing of users' microblog streams: the results may be out-of-date when they finally arrive. Also, this centralized data processing affects the scalability of the spam detection process. Second, they mainly focus on offline historical logs [29], [32], [45], [73], [85], which may not promptly generate updated detection models to discover the latest spam. Third, they ignore the spam correlations among different data sources [40], [77], [86], limiting their capability to leverage broad trends to enhance the accuracy of spam detection. Recently, Twitter and Facebook deployed a new feature that can display labels for misinformation and rumors [17]. However, they still mainly rely on fact-checking from subject-matter experts and are limited to single data sources [16].

In this paper, we propose a novel online scalable spam detection system, namely `Spiral`. The key goal of `Spiral` is to advance the general spam detection processing by enhancing the **online** and **scalability** features while enabling **spam correlation** among distributed sources.

This paper makes the following contributions:

- An architecture to achieve information integrity, i.e., an **online** spam detection system, called `Spiral`, to defend against real-time spam activities that happen in geo-distributed sites.
- A peer group communication structure that allows multiple social network domains to exchange and utilize **spam correlations** to enhance the accuracy of spam detection.
- A comprehensive evaluation of Spiral's performance and functionality on a large cluster using real-world social stream data, demonstrating the feasibility of our approach.

Experimental results show that `Spiral` achieves good performance, scalability, and online spam detection accuracy. It achieves millisecond-scale latency in data delivery, aggregation, and group communication. It exhibits low runtime overhead. It can balance workloads and scale to hundreds of thousands of agents that process millions of spam posts concurrently. We compare `Spiral` with state-of-the-art spam detection methods (Bi-gram&C [75] and CBFs [56]), showing that `Spiral` achieves superior performance without accuracy loss with F1 scores of up to 97%.

The remainder of this paper is organized as follows. Section II discusses the related work. Section III introduces the background. Section IV describes the `Spiral` design and implementation. Section V shows the experimental setup and performance evaluation results. We conclude with some directions for future work in Section VI.

II. RELATED WORK

There is a significant body of work on spam detection in online social networks. We focus here on the most related solutions, models, and systems. `Spiral` introduces a novel decentralized system architecture that leverages stream processing, a consistent DHT-based ring with routing, and publish/subscribe trees. As such, `Spiral` significantly expands upon the state-of-the-art approaches.

A. MAINSTREAM METHODS FOR SPAM DETECTION

Existing efforts mostly focus on offline methods in analyzing social data [29], [32], [37], [45], [52], [55], [85]. For example, by analyzing the historical log data, some methods use crowd signals (such as users' flagging accuracy [52], post contents and user comments [68], account features like followers relationship [29], and extracted text from photos [37]) as the set of features of the spam classifier. Many recent projects

leverage advanced natural language processing models (e.g., deep learning models) to identify spam contents and activities by using propagation pathways of posts [76], users-content interaction graph [45], users' meta-data [32], and utilizing user attributes, content, activities and relationships [85]. However, most of these features can only be collected after spam has circulated for a while, having already reached and been responded to by many users. Moreover, spammers are good learners in adjusting their activities or behavior patterns to escape being detected from former features. Thus, studies limited to historical data sets have difficulty catching up with the newly emerged spam posts in real time. Recently, Twitter and Facebook deployed a new feature that can display labels for misinformation and rumors [17]. For example, Twitter uses three levels of warning labels – ''Get the latest'', ''Stay Informed'', and ''Misleading'' to offer more information [18]. However, they still mainly rely on fact-checking by subject-matter experts and limit their analysis to a single data source [16].

B. SPAM DATA PROCESSING PIPELINE

Existing systems for detecting spam campaigns [37], [46], [64], [83], [87] mostly follow the classic big data analytics pipeline: all user's microblog streams are first collected by (geographically) distributed log collection systems (e.g., Flume [2], LinkedIn's Kafka [8], Yahoo's Chukwa [62]), and then transferred to data storage (e.g., MongoDB [9], HDFS [4], HBase [5]) in a central data center, and then processed with engines such as Spark Streaming with MLlib [11] and GraphX [48]. However, spam dissemination patterns have been continuously evolving. Due to the long delay introduced by log collection and log processing, these systems fail to adapt to emerging spam methods. Moreover, existing stream processing systems (e.g Spark [82], Storm [7], Kafka [8], Flink [1], Samza [6], Wukong+S [84], Pregel [54], Giraph [3]) are designed for general-purpose stream processing tasks such as climate modeling, web crawling, targeting advertising, and datacenter intrusion detection, which often do not have the strict real-time and scalability requirements required by spam detection systems. They employ a centralized ''one master/many slaves'' architecture. In such an architecture, ''only one'' master is responsible for administering many jobs (suppose the spam detection for each social event is a job), including scheduling each job's tasks to different machines, monitoring each job's progress, tracking state snapshots, and handling failures and stragglers. As a result, the central master easily turns into a central bottleneck and single point of failure when it is necessary to handle numerous spam instances on many newly emerged posts simultaneously.

C. CORRELATIONS BETWEEN MULTIPLE DATA SOURCES

To our knowledge, there are few studies broadly relevant to correlations between feature-similar social networks or social activities. Existing work either points out the criticality of spam correlations [30], [35], [50] or develops mechanisms for detecting these correlations [40], [78], [86]. These studies propose no effective solutions for leveraging correlations for spam detection. Cinelli et al. [40] argue that reliable or questionable data sources have no significant differences in the spreading patterns, according to their analysis of data from multiple social platforms. Curtis et al. [50] point out that the features of social bots can easily be applicable across different social platforms and they use similar algorithms to detect social bots. Wu and Zhou [78], [86] show that social influence is a common phenomenon in social networks where highly connected users or central users of one group/activity are the core content disseminators, and retrieval of knowledge and propagation patterns from one platform can help the analysis in another platform. None of these investigations addressed how to achieve correlation in real-time and at the tremendous scale required by widely adopted social networks.

III. SPIRAL's MOTIVATING BACKGROUND

The key insight in our approach is to re-architect spam detection systems using a decentralized system model.

A. PEER-TO-PEER MODELS

The P2P model has been widely used in BitTorrent [60], Bitcoin [34], mobility management in 5G network [20], and blockchain [21]. Unlike the traditional client/server model, in which clients make service requests and servers fulfill them, the P2P model allows each node to function as both a client and server [25]. In the P2P model, each node is equal to the other nodes, and they have the same rights and duties. The primary purpose of the P2P model is to enable all nodes to work collaboratively to accomplish a task or deliver a specific service. For example, in BitTorrent [60], if someone downloads a file, the file is downloaded to her computer in bits and parts that come from many other computers in the system that already have that file. At the same time, the file is also sent (uploaded) from her computer to others who ask for it. Similarly to BitTorrent, in which many machines work collaboratively to undertake the task of downloading and uploading files, we enable all distributed nodes to work collaboratively to undertake the duty of scalable online spam detection over geo-distributed data sources. Pastry [66] and Chord [71] are widely adopted realizations of the P2P model. Note that P2P models are used for file sharing and distributed data storage. To the best of our knowledge, we are the first to leverage P2P technologies to re-architect the online social spam detection systems.

Figure 2 details a BitTorrent use case to illustrate how services are distributed across nodes with a peer-to-peer model. When user A submits the request for downloading a specific File B, that download request is forwarded to distributed clients in the peer-to-peer network (i.e., a network where all nodes communicate with each other). A node that has some part of the file B data will respond to the request and send some of its B file data to user A. User A can grab multiple parts of File B from multiple distributed

sources and simultaneously download those different bits, then putting the file together like a jigsaw. Meanwhile, User A's node can also serve other clients, providing data to fulfill requests from other nodes. The peer-to-peer network can provide quick file searches and high-speed file downloads. P2P models eliminate the long latencies present in centralized systems, where files are downloaded directly from a single remote server which quickly becomes a bottleneck.

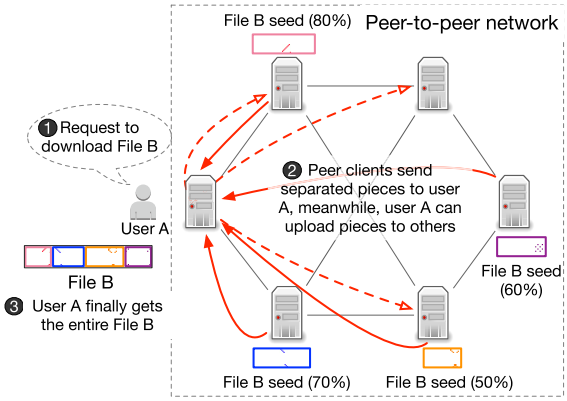


FIGURE 2. An example of file downloading in the BitTorrent peer-to-peer network. It illustrates that participant nodes are both clients (asking for file data) and servers (storing and providing file data).

To realize a P2P model, we utilize distributed hash tables (DHTs). A DHT is a distributed key-value store architecture that follows the P2P paradigm. The key-value pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. The main advantage of DHTs is their flexibility: keys can be redistributed with minimal work to add or remove nodes to the system, which allows a DHT to easily scale to extremely large numbers of nodes and handle continual node arrivals, departures, and failures [22]. The DHT-based network overlay is built by performing virtual tunnels between the participating node. Typically, a tunnel of a pair or nodes will traverse multiple tunnels in the underlying network. In such an overlay of N nodes, the paths of the tunnel are maintained as $O(logN)$ depths by following the DHT protocol. A DHT-based overlay can support scalability, fault tolerance, fast searching, correctness under concurrency, and load balancing [49].

B. DECENTRALIZED OVERLAY AND MESSAGE ROUTING

Inspired by the success of DHT-based decentralized overlay in peer-to-peer networks [38], [66], `Spiral` uses DHT-based routing protocols to (1) create multiple DHT-based functional trees, where each tree constitutes a `Spiral` group to feed the distributed online social data sources into a decentralized peer-to-peer overlay, (2) instantly update the detection classifiers and exchange newest spam information with all peer-groups, and (3) disseminate the updated spam models to all distributed agents in a timely manner.

`Spiral` leverages Pastry [66] to build a DHT-based overlay. In Pastry, each node is assigned a `nodeId` (128 bits) used to identify nodes and route messages. 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 $b = 4$), where N is the number of nodes). Each node maintains a routing table to support the message routing, self-organization, and fault recovery functionality.

At each routing step, given a key, Pastry routes messages to the node whose `nodeId` is numerically closest to the key. The node first checks if the key falls within the range of the `nodeIds's` leaf set. If so, the message is directly forwarded to that node. If not, the message is forwarded to another node in the routing table whose `nodeId` shares a common prefix with the key by at least one more digit. In some cases, there is no appropriate entry in the routing table, or the associated node is not reachable. Then the message is forwarded to a node whose prefix is the same as the local node but numerically closer [66].

C. GROUP MANAGEMENT

`Spiral` leverages Scribe [38] to construct the application-level group communication system. Scribe can easily and effectively handle a group from one to millions of group members (nodes), and all nodes can flexibly join and leave groups [38]. It names a group by a pseudo-random Pastry key, which is called `groupId`. Typically, the groupId is the hash of the group's textual name concatenated with its creator's name. A node routes a CREATE message by using the `groupId` as the key to create a new group. The node responsible for that key becomes the manager of the group. To join a group, a node routes a JOIN message towards the `groupId`. The message will continue to be routed until it reaches a node in that group. Scribe can efficiently support large numbers of groups with highly dynamic memberships.

`Spiral` group supports two major functionalities: *multicast* and *anycast*.

- *Multicast* is used to construct a hierarchical aggregation tree, which is a fundamental `Spiral` abstraction. *Multicast* allows messages or instructions to be delivered to all the members with very low latency. Any nodes in the overlay can create a group; other nodes can join the group and then *multicast* messages to all members of the group along the tree.
- *Anycast* is used for group communication and model transmissions among multiple groups. It is implemented by using a distributed depth-first search (DFS) of the functional tree. Each node in the overlay (which may be in or out of group k) can *anycast* to the group k by routing a message towards the group k's `groupId`. The local route convergence in Pastry guarantees that this message will reach a group member near the sender's `nodeId`. *Anycast* can also be used for communication between multiple groups, such as exchanging updated data and spam detection models.

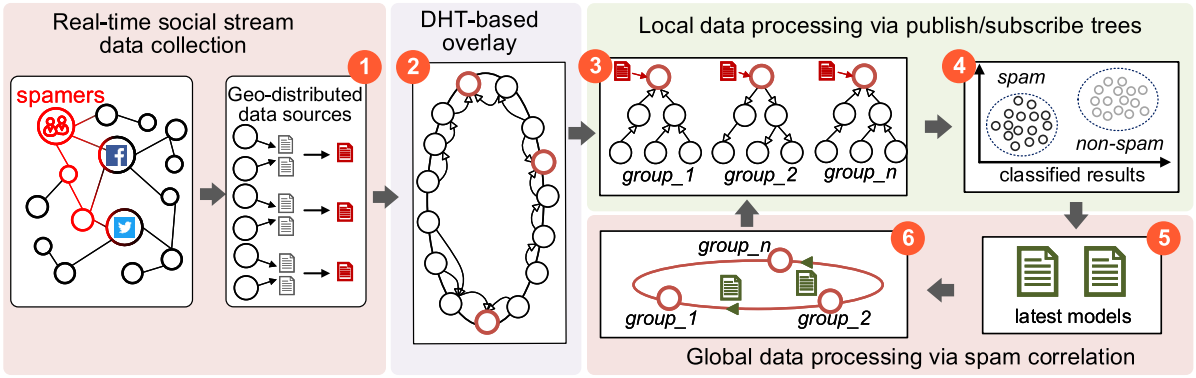


FIGURE 3. The `Spiral` workflow. First, Spiral deploys hundreds of thousands of agents and these agents collect the training data and live stream data from geo-distributed sites. Second, it organizes the agents into a DHT-based ring overlay. Third, agents can join many peer-to-peer functional Scribe trees; each tree is a `Spiral` group, which consists of a root, branches, and many leaves. Fourth, a single group completes its local data processing and outputs the latest spam. Then outputs are aggregated into the root. Fifth, the roots of many groups instantly update their spam detection models based on the latest social spam. Finally, roots exchange their newest models with the other peer groups.

IV. SYSTEM DESIGN AND IMPLEMENTATION

In this section, we introduce the `Spiral` system, outline workflow details, and discuss each of its components.

A. OVERVIEW

`Spiral` operates in four phases. The first phase is **real-time social stream data collection**. `Spiral` consists of a large number of distributed agents that reside in geo-distributed sites, which gather the real-time social data streams through the social networks' public APIs (e.g., Twitter streaming API [14], Facebook Graph API [13]) in a scalable way. The second phase optimizes data dissemination by leveraging the efficiency and scalability of distributed hash tables (DHTs) through a **DHT-based overlay deployment**. By using the Spiral system, decentralized communication of spam trends occurs without developers having to worry about distributed computing. The third phase is **local data processing via publish/subscribe trees**. All leaf agents of the `Spiral` group in a local social media site take the new incoming data streams (Twitter logs, Facebook logs, and the like), analyze them using the spam detection model trained from the first phase, and output labels. The posts labeled as spam are then aggregated to the root and reported to the end users. The fourth phase is **global data processing via spam correlations**. After many `Spiral` groups complete their local data processing, the roots instantly update their spam detection models based on the latest data traffic, then disseminate to other peer groups so that each group can have a global view of the correlated social spam from geo-distributed data sources. Each group keeps the latest models so that it can utilize the spam correlations and process the new incoming online social data streams from a global perspective.

B. WORKFLOW DETAILS

Figure 3 shows the `Spiral` workflow.

Next, we discuss the details of each phase.

- **Phase 1: real-time stream data collection**. `Spiral` collects two types of data: (1) training data for creating the spam detection model and (2) live social stream data. To collect the training data, we manually extract the dataset and label them. This training dataset is used to create a spam detection model. To collect the live social stream data, as shown in the first step of Figure 3, `Spiral` deploys hundreds of thousands of agents in geo-distributed sites, each of which is associated with a web server that collects real-time social stream data via social networks' open APIs (e.g., Twitter streaming API [14], Facebook Graph API [13], etc.) More details can be found in subsection IV-C.
- **Phase 2: DHT-based overlay deployment**. The agents in Phase 1 are organized into a DHT-based ring overlay [66]. Agents can join many peer-to-peer functional Scribe trees [38] and each tree is a `Spiral` group. Each `Spiral` group consists of a root, branches, and many leaf agents. The root deploys the spam detection model (for example, using Random Forest [39], [41] or Naive Bayes [39], [69]) based on the training dataset from Phase 1. This model is used in the local data processing in a single group (details in subsection IV-E) and global data processing in many groups (more details can be found in subsection IV-F).
- **Phase 3: local data processing via publish/subscribe trees**. The local data processing refers to the spam detection in a single `Spiral` publish/subscribe tree. The input of this processing is the spam detection model from Phase 2 and the live stream data from Phase 1. As shown in the third and fourth steps of Figure 3, first, the group's root disseminates the spam detection model to its following agents via the functional tree. Then the leaf agents in each group apply the spam detection model to process the live stream data and output the spam with labels. The output of the local spam detection is the pairs of posts with labels. The result pairs are aggregated into the root agent, which filters out the latest spam and uses them in the global data processing (more details can be found in subsection IV-E).

- **Phase 4: global data processing via spam correlations**. As illustrated in the fifth step of Figure 3, after Spiral groups complete their local data processing, the roots instantly update their spam detection models based on the latest social spam, and then exchange their newest models with the other peer groups. As shown in the final step of Figure 3, all groups now have a global view of the correlated social spam from geo-distributed data sources. Each group uses the latest model to start a new round of local data processing. By leveraging spam correlations, they can keep pace with the latest spam information from a global perspective (more details can be found in subsection IV-F).

C. REAL-TIME STREAM DATA COLLECTION

The data collection in Spiral consists of two parts: (1) training data collection and (2) real-time stream data collection from geo-distributed sites. Next, we present details of these two kinds of collections.

Training data collection refers to the creation of training datasets. A critical challenge is how to create an appropriate initial dataset that has confirmed social spam. The amount of real-time social media content is huge, e.g., around 350,000 tweets are sent per minute on Twitter [15]. Spam posts are only a tiny portion of the total social media content. In order to avoid collecting malicious posts by looking for a "needle in a haystack," we narrow down the scope of the collection by leveraging real spam posts and focusing on capturing how they spread. Fortunately, there exist several online debunking websites (such as Snopes.com, PolitiFact.org, and FactCheck.org) that allow us to check the credibility of a specific post. After collecting posts from social media, we can use the information from debunking websites to label our dataset and classify which posts are spam, and format the information as the original training dataset. We mark a post as *spam* if there is strong or straightforward evidence in URLs or contents, e.g., malicious information, fake advertisements, and URL links to non-credible websites [57]. The training dataset can be divided into two sets for training and testing the spam detection model.

Real-time social stream data collection refers to the online data collection from geo-distributed sites. All leaf agents carry out this collection. Each leaf agent is associated with a web server that continuously produces online social stream data (e.g., tweets, posts, hashtags, and news). The leaf agent uses social media open APIs to collect the online/real-time streaming social logs. For example, Twitter posts can be collected from the tweet streams that cover worldwide posts, and Facebook posts are collected from open accounts and public pages. Further, the leaf agent normalizes the raw social stream data into the same formatted *stream datasets*, for example, keeping the contents under the same maximum length and maintaining the top three hashtags. Note that leaf agents can set different time windows for log collection or search for different types of logs by keywords or timelines [13], [14].

D. DHT-BASED OVERLAY DEPLOYMENT

In this subsection, we detail the process of building a DHT-based functional tree and introduce its roles.

1) DHT-BASED TREE CONSTRUCTION

For the Spiral's tree construction, the first step is to construct a peer-to-peer overlay by leveraging Pastry [66]. Each Spiral agent has a unique, 128-bit nodeId in a circular nodeId space ranging from 0 to $2^{128} - 1$. Note that the 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 $O(logN)$ steps. The second step is to build a hierarchical tree by leveraging Scribe [38]. Any node in the overlay can create a group with a groupId that is the *hash* (SHA-1 [10]) of 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 node whose nodeId is numerically closest the groupId will serve as the root.

2) FUNCTIONAL TREE

The Spiral functional tree is responsible for creating efficient paths for the root agent to disseminate models to the leaf agents. The key idea is the use of a DHT-based application-level multicast [38], similar to the IP multicast [44], to propagate models following the tree path without maintaining N point-to-point connections for N leaf agents. As illustrated in Figure 4, assuming there are seven distributed agents jointly working in group "*video*", the agent with nodeId numerically closest to the groupId (*hash(video + creator name)*) acts as the rendezvous point for the functional tree. For example, if *hash*(video + creator name) equals to *EA34*, the agent with the closest nodeId like *EA34* or *EA35* will serve as the root. The tree is rooted at the rendezvous point and the other agents can subscribe to this tree. The root can multicast messages, instructions, or models to all leaf agents in $O(logN)$ hops.

Note that Spiral's tree structure can be self-tuned to satisfy different latency requirements. For example, if the primary goal of a user-defined application is low latency, Spiral can customize the depth of the functional tree by adjusting the value of the fan-out parameter n. Assuming there are 10^b agents in the system, the default depth of the tree is $log_{2^n}N$, where N is the number of agents. The average depth of the functional tree can be pruned from 5 to 4 by altering the default fan-out parameter from 4 to 5 (changing the logarithmic base from 2^4 to 2^5). Therefore, the average delivery latency from root to leaf can be reduced by 20%.

3) DATA COLLECTION VIA FUNCTIONAL TREES

The collection of relevant social data is fulfilled by the leaf nodes in the functional trees. We adopt two policies for the leaf nodes' data collection. First, with the high flexibility and easiness of social platforms' APIs (Application Programming

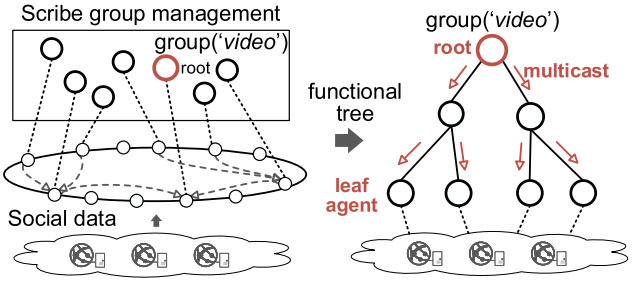


FIGURE 4. `Spiral` **group management and functional tree. The functional tree is responsible for creating paths and orchestrating distributed agents, e.g., paths for root multicasting models and aggregating results. The multicast processing can be finished within** ***O(logN)*** **hops.**

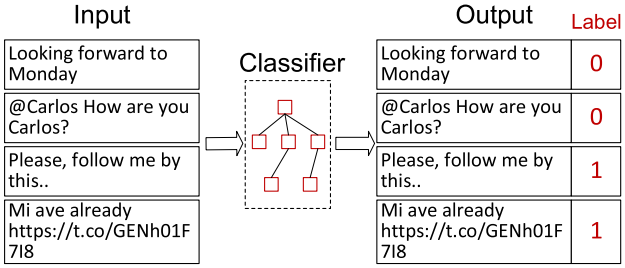


FIGURE 5. Example of local data processing in a leaf agent. The original data has no labels. After classification, the predicted labels are generated. Label 1 represents ***Spam*** **and 0 represents** ***Ham*** **(non-spam).**

Interfaces), each leaf node can maintain the connection of the API to collect the real-time streaming social data. For example, Twitter APIs [26], YouTube APIs [28], and LinkedIn APIs [24] provide plenty of functionalities that allow to easily collect the relevant social content via the timeline, keywords, or specific hashtag. In this case, each leaf node can maintain access to APIs and instantly collect the targeted data by following the user requirements such as defined keywords or specific time periods. Second, several social platforms' APIs are not as open as previous platforms, such as Facebook Graph APIs [13] and Instagram APIs [23], which have some limits due to their privacy policies and scope permissions. Under this circumstance, leaf nodes can first seek to get a permanent access token, called long-lived tokens. This allows the collection of long-running jobs could be more flexible in future processing. Besides, leaf nodes can work in a group to collect the relevant social data if some APIs have data volume limits per minute or hour.

The raw social data collected is first normalized to a JSON file, as data from the social platform API is in JSON format by default. We set a default threshold for raw file sharding (this threshold can be refined based on application characteristics or user requirements). When the threshold is reached, the raw data file will be pre-processed by the root node in the functional tree and the data analysis work will begin. The root node supports functions such as data cleaning, including removing duplicate or irrelevant data, debunking URLs, fixing syntax errors, etc. When the pre-processing data set is ready, the functional tree will work according to the local and global data processing pipeline.

E. LOCAL DATA PROCESSING VIA PUBLISH/SUBSCRIBE TREES

Local data processing refers to the spam detection within a single group that makes decisions on each incoming social post by marking it as either spam or legitimate. Figure 5 presents an example of spam detection in a leaf agent. After the spam detection model categorizes the log, an identified label (here 1 represents spam and 0 represents non-spam) is created for each post in the original social logs. This spam detection decision is made by a trained classifier from the supervised machine learning module. The pairs of posts and associated labels are aggregated by the root agent. The `Spiral` tree progressively rolls up the intermediate results (i.e., post-label pairs) and reduces them from the distributed leaf agents to the root. For example, as shown in Figure 6, $\langle(788A, 1), (2D17, 0)\ldots\rangle$, $\langle(0DA4, 0), (788A, 1)\rangle$, are reduced to $\langle(788A, 2), (2D17, 0), (0DA4, 0)\ldots\rangle$ in the branches of tree. These pairs are then reduced to $\langle(788A, 3), (D1C6, 1), (2D17, 0), (0DA4, 0)\ldots\rangle$ when they arrive at the root. The label values indicate the number of leaf agents that identified the post as spam. For example, $(788A, 3)$ represents that 3 leaf agents have classified the data "*mnk*" as a spam post. Larger values indicate a higher probability of a post being spam.

The criteria for spam classification are based on URL and content features. For the URL features, spammers typically obfuscate malicious URLs by adding spaces and Unicode characters into them [47]. This technique is a simple but effective way to bypass the filters that block URLs. Inspired by [47], we clean URLs by inverting the obfuscation process, including removing white space padding and normalizing URL encoded characters (e.g., "*subsexvideo%26ip%3Dauto%26click%3D1*" becomes "*subsexvideo&ip=auto&click=1*"). For the content features, we remove punctuation, convert all letters to lowercase, tokenize each word, remove stop words (i.e., articles, prepositions, pronouns, conjunctions, etc.), and do stemming (i.e., reduce inflected and derived words to a root form) using the Porter stemmer [59]. We extract the *term frequency* and *inverse document frequency* (tf-idf) values of the terms in each document. The tf-idf yields a weight that measures the importance of a term in the corpus for its post [61].

F. GLOBAL DATA PROCESSING VIA SPAM CORRELATIONS

The global data processing is completed through peer group coordination, which shares the latest spam detection models among groups. Once a group finishes its local data processing, the root agent aggregates the results containing the newest spam information and then updates its local training dataset and spam detection model. Next, the root agent exchanges the latest model with other peer groups so that all groups can have the latest models to enhance the spam detection accuracy rate.

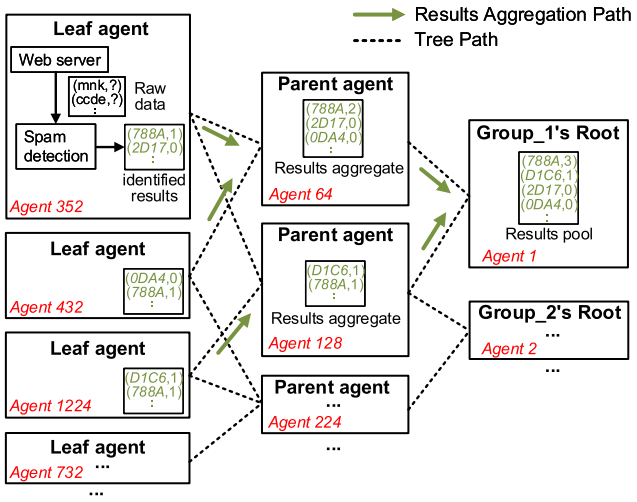


FIGURE 6. Aggregation in the `Spiral` functional tree. Each leaf agent produces the classified results from collected social streaming data via web servers. Then it sends the results via the aggregation path to the upper layer until they reach the group root's result pool.

We design a diffusion group in `Spiral` to minimize the latency of group coordination. Each root agent holds a model table that contains the model version and the original `groupId`, denoted as a <`groupId`, *versionNum*>. The root within the updated model in one group randomly chooses two other groups to disseminate the model with the same version number. For example, when group "*video*" builds its latest model, it selects two groups "*picture*" and "*link*" as targets and delivers its model with the instance <*video*, *08*> to these groups. Here *08* is the current *versionNum* of the latest model. The roots in group "*picture*" and group "*link*" then check their model tables to see if the received model is the latest. If not, they update the model to the latest version and act as new communicators to disseminate the latest model to other groups. Eventually, all groups have the latest model. The number of rounds to propagate a single latest model to all groups is $O(log(m))$, where m is the number of diffusion groups.

G. ANALYSIS

In this subsection, we leverage existing peer-to-peer analytical work [58], [63] to design a performance model for `Spiral`: *How long does it take to deliver a spam detection model from root agent to its leaf agents and aggregate back classification results from leaf agents to root?*

In general, a peer forwards its data to any other peers and repeats this process until the number of hops reaches a specified threshold θ. The value of θ is the average depth $(log(N))$ of the functional tree that consists of N agents. Therefore, the model delivery time and the result aggregation time mainly depend on the number of routers in the path that social stream data has been delivered. The peer-to-peer overlay network consists of various links and relevant routers, so that the overall number of routers shall not be the same as the calculated hops. Therefore, in this overlay network, to define the average number of routers between two agents, we use the expected shortest path between two random agents. By using the random graph in the overlay network [58], the distance can be approximately expressed as:

$$d = \frac{\ln[(N_r - 1)(z_2 - z_1)] - \ln(z_1^2)}{\ln(z_2/z_1)} \tag{1}$$

where N_r is the number of agents in the router graph and z_i is the average number of i_th hop's neighbors in the graph. z_2 and z_1 can be expressed by using the router adjacency matrix.

We next define the network latency in the router. The expected waiting time of the i^{th} router can be expressed as:

$$E(w_i) = \tau_i \rho_i g_i (c_{ai}^2 + c_{sj}^2)/2(1 - \rho_i) \tag{2}$$

where τ_i is the mean time for the router to process a packet, ρ_i is the traffic intensity in the router i, and c_{ai}^2 and c_{si}^2 are the squared coefficient of variation (SCV) of the arrival process and service time at router i. Then $g_i \equiv g_i(\rho_i, c_{ai}^2, c_{sj}^2)$ can be defined as:

$$g_i(\rho_i, c_{ai}^2, c_{sj}^2) = \begin{cases} 1 & c_{ai}^2 \geqslant 1 \\ exp[-\frac{2(1-\rho_i)}{3\rho_i} \frac{(1-c_{ai}^2)^2}{c_{ai}^2 + c_{sj}^2}] & c_{ai}^2 < 1 \end{cases} \tag{3}$$

The delivery of models is completed within θ hops with an average of d routers, and as the return path, the aggregation of results has an equal number of routers. They are determined by the slowest agent in the functional tree who finalizes the delivery and aggregation. Therefore, the total expected time cost can be expressed as:

$$E(T) = \max_N \left(2\theta d \sum_{i=1}^{N_r} E(w_i)\right)/N_r \tag{4}$$

The above analysis provides us a way to analyze the expected performance of `Spiral` in terms of the physical network environment, which can be used to optimize the utilization of hardware resources and the system deployment in real-world scenarios.

V. EVALUATION

We evaluate the `Spiral` system by using real-world online social network streaming data. We have collected 3,000,000 tweets from Twitter streaming APIs from January 2017 to April 2017 and nearly 1,000,000 posts from Facebook APIs from March 2017 to June 2017. We manually labeled and created the training and test datasets. These datasets were labeled based on the posts' URL, content, and official social media identifications. Experiments were conducted on a testbed of 10,000 agents hosted by 20 Linux 3.10.0 servers. Each server has a Intel four-cores CPU with a 3.4 GHz processor, 4 GB of memory, and 30 GB hard drives. The system is implemented using Java version 1.7.

The experimental evaluation consists of three parts. First, we evaluate the spam detection accuracy rate of `Spiral`

TABLE 1. Local data processing results.

Model	F1	Precision	Recall
Random Forest	0.962	0.962	0.962
KNN	0.911	0.902	0.916
Logistic	0.908	0.911	0.905
Naïve Bayes	0.871	0.886	0.856
Random Tree	0.863	0.876	0.850

TABLE 2. Global data processing results.

Model	F1	Precision	Recall
RF	0.973 (1.1% ↑)	0.973 (1.1% ↑)	0.973 (1.1% ↑)
KNN	0.920 (0.9% ↑)	0.925 (2.3% ↑)	0.920 (0.4% ↑)
Logistic	0.942 (3.4% ↑)	0.944 (3.3% ↑)	0.942 (3.7% ↑)
NB	0.931 (6.0% ↑)	0.932 (4.6% ↑)	0.931 (7.5% ↑)
RT	0.936 (7.3% ↑)	0.937 (6.1% ↑)	0.935 (8.5% ↑)

RF: Random Forest; *NB*: Naïve Bayes; *RT*: Random Tree.

and compare it with the state-of-the-art work (details in subsection V-A). Second, we evaluate the performance of the Spiral functional tree in terms of scalability, latency, load-balance, and system reliability (details in subsection V-B). Third, we evaluate the runtime overhead of the system (details in subsection V-C).

A. SPAM CLASSIFICATION RESULTS

1) LOCAL SPAM DETECTION

We first evaluate the performance of local spam detection. Here local spam detection refers to local processing in a single group without coordinating with other groups (i.e., updated models are not exchanged with other groups). We implement several algorithms (Random Forest (RF) [36], KNN [53], Logistic [51], Naive Bayes (NB) [65], and Random Tree (TR) [31]) on Spiral and evaluate them on a dataset that consists of 50,000 posts (half posts are spam). We use the performance metrics of *F1 score*, *precision*, and *recall*, where *F1 score* is the harmonic mean of the precision and recall, *precision* is the fraction of correctly identified posts in the predicted labels, and *recall* is the fraction of correctly identified posts in the original labels. As illustrated in Table 1, the Random Forest (RF) algorithm achieves F1 scores of up to 96%, demonstrating Spiral's ability to detect spam posts with a high accuracy rate while achieving scalability.

TABLE 3. The comparison of Spiral and the state-of-the-art models.

system (model)	F1	Precision	Recall
Spiral (RF)	**0.973**	**0.973**	**0.973**
Spiral (Logistic)	0.942	0.944	0.942
Bi-gram&C (RF)	0.969	0.970	0.969
Bi-gram&C (Logistic)	0.947	0.950	0.949
CBFs (RF)	0.965	0.966	0.965
CBFs (Logistic)	0.959	0.958	0.959

2) GLOBAL SPAM DETECTION

Spiral seeks to enhance the spam detection accuracy rate by leveraging the correlations between different data sources. Spiral groups exchange the newest models with each other to ensure that every group catches up with the newest spam information. As illustrated in Table 2, we can observe that by using the spam correlations, the final detection rate (F1 score) achieves increases of 1% to 7%. This data illustrates that spam correlations can improve spam detection performance.

We compare Spiral's spam classification results with other popular spam detection models (Bi-gram&C [75] and CBFs [56]). We reformat the data by extracting the content features according to the models in [56] and [75]. As shown in Table 3, Spiral's F1 score goes as high as 97%. This demonstrates Spiral's ability to detect spam posts with high accuracy rate.

B. SPIRAL FUNCTIONAL TREE PERFORMANCE

In this subsection, we evaluate the performance of the Spiral functional trees in terms of delivery latency, system reliability, and scalability. Specifically, we focus on the model delivery latency, result aggregation latency, tree construction latency, failure recovery latency, and load balance.

We evaluate Spiral using 1,000 to 10,000 agents and create 10 functional trees. To show the effect of the tree structure on the latency, we set up trees with different tree bits (i.e., tree fan-out) in the system, where the larger tree bit indicates that the tree has a larger fan-out and fewer layers.

1) TREE CONSTRUCTION TIME

We evaluate the tree construction time by varying the number of agents and tree structures. As shown in Figure 7a, the tree construction time linearly increases with the number of agents. When using different tree structures (different tree bits), the latency of tree construction is quite consistent, showing that Spiral has relatively stable performance under different tree structures.

2) MODEL DELIVERY LATENCY

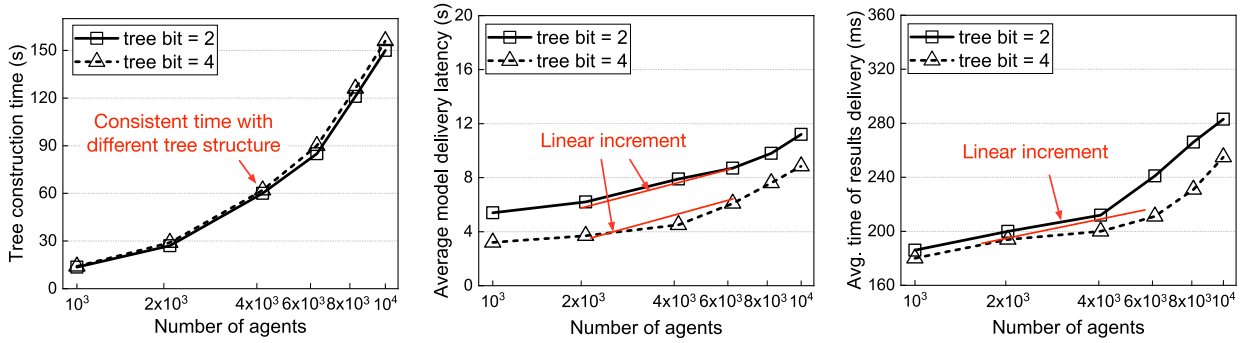
Figure 7b shows the model delivery latency when using a Random Tree model with 5k data blocks (a training dataset has 5,000 instances). Results show that even if the number of agents is multiplied, the model delivery latency (root-to-leaf) increases linearly. This scalability is achieved because the increment of the model delivery latency is strictly determined by the tree depth $O(logN)$.

3) RESULT AGGREGATION LATENCY

Figure 7c shows the average latency for aggregating results from a leaf agent to the root. The path of result aggregation is following the tree so that the tree's structure directly influences this latency. Results show that the aggregation latency from leaf to the root increases linearly when doubling the agents. It demonstrates the efficiency of the functional tree in completing the result aggregation from leaf agents to root.

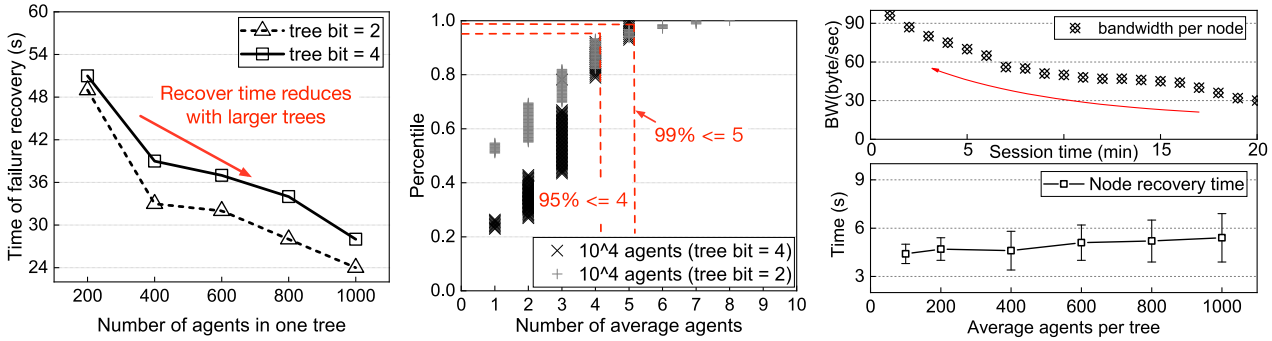
4) FAILURE RECOVERY LATENCY

The recovery process is as follows: when an agent fails, it notifies all members of its leaf set. After a new agent is



(a) Tree construction time. (b) Model delivery latency. (c) Result aggregation latency.

FIGURE 7. Performance evaluation of `Spiral` in terms of tree construction time, model delivery time, and spam classification results delivery time. (a) The tree construction time for different numbers of agents. (b) Time for leaf agents to receive models from the root agent. (c) Time of shuffling data from leaf agents to the root agent in the tree.



(a) Failure recovery latency. (b) Scalability. (c) Agent churn.

FIGURE 8. Performance evaluation of `Spiral` in terms of the failure recovery latency, percentiles of average agents involved in result aggregation, and agent churn. (a) The average failure recovery latency of different tree structures. (b) The distribution of participating agents in the result aggregation. (c) The performance of agent churn in the system.

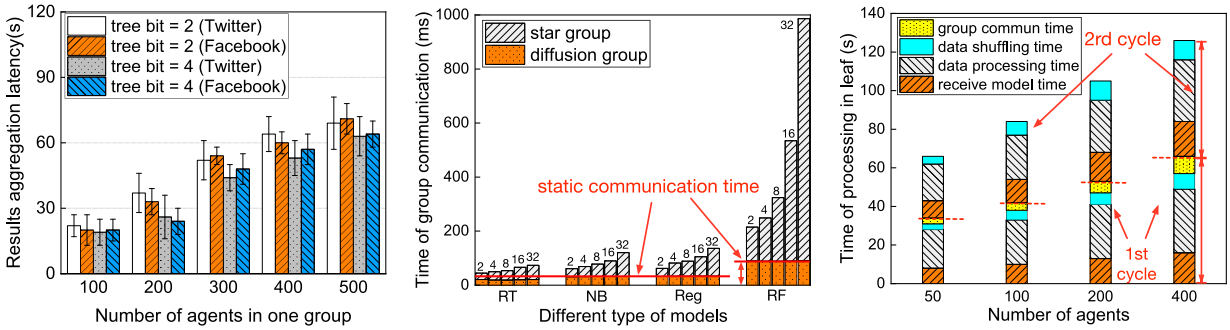
restored, it contacts the agents in its last known leaf set, gets its current leaf set, updates its own leaf set, and then notifies the members in its new leaf set of its existence. Therefore, the fan-out has a significant impact on the failure recovery time. Figure 8a shows the relationship between the failure recovery time and the tree structure. Results show that the failure recovery time increases as the tree bit increases because when a tree has a larger fan-out, it has more children agents. Once an agent fails, all sub-agents need to restore their functions and find new parents, costing more time for failure recovery. Besides, as the number of agents increases, the tree becomes larger and the recovery latency decreases. This decrease happens because as the tree gets deeper, the impact of an agent's failure is reduced and other agents can quickly route to new parents.

5) SCALABILITY

To assess `Spiral`'s scalability, we follow a result aggregation scenario with 2,000 pairs (spam and its label) and track the intermediate agents involved in forwarding and delivering. Figure 8b shows the percentiles of average participating agents for one result pair's aggregation. We can observe that 95% of aggregations have no more than four processing agents when the tree bit is 2, and 99% of aggregations have no more than five working agents when the tree bit is 4. These results demonstrate Spiral's load balance capabilities when deploying a large number of aggregated processes.

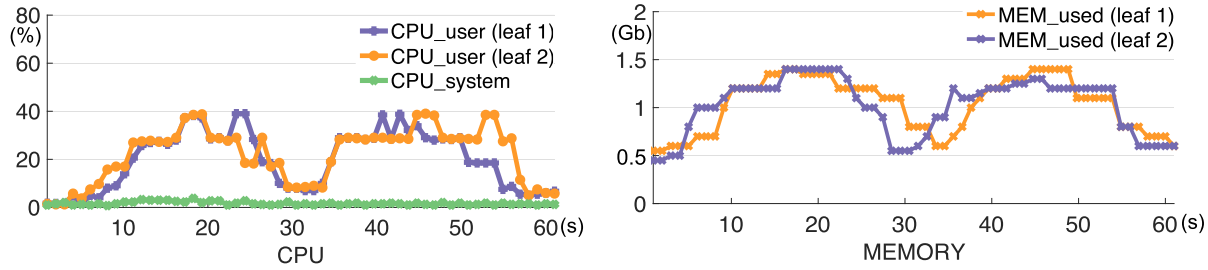
6) CHURN OF AGENTS

We evaluate the performance of the churn of the node. Agents (nodes) in the network can dynamically join and leave a group, which may cause node churn. Figure 8c shows the performance of agent churn in the system, where we simulate the group session time for one agent from 1 minute (high churn) to 20 minutes (low churn). We can observe that when nodes are highly fluctuating (high churn), the performance of the system is faster exacerbated. Generally, agents in a group are not required to frequently leave or join the group, the session time can be determined by the features of specific data applications. Therefore, the DHT-based overlay in `Spiral` is churn-resilience due to the low churn. Besides, the results of recovering churn nodes show that with different sizes of functional trees, churn nodes can be quickly recovered in the overlay, ensuring sufficient availability for various social data applications.



(a) Group aggregation latency. (b) Group coordination time. (c) Time of various processing parts.

FIGURE 9. Performance evaluation of `Spiral` in terms of the average latency of result aggregates in one group, group communication time, and time of various processing parts. (a) The average latency of root agent aggregates intermediate results from Twitter and Facebook data streams in one processing cycle. (b) The group coordination time of sharing models with diffusion group and star group in `Spiral`. (c) The processing time of various processing parts with two cycles of processing in one group.



(a) CPU utilization of one agent. (b) Memory utilization of one agent.

FIGURE 10. Runtime overhead.

7) GROUP AGGREGATION LATENCY

Figure 9a shows the latency of result aggregation in one group by varying the number of agents (from 100 to 500). Differently from Figure 7c, which displays the latency of aggregating results from only one leaf agent, Figure 9a shows the entire aggregation latency from all leaf agents. It shows that the aggregation latency increases linearly as the number of agents in a group doubly increases. This data indicates that the `Spiral` group can quickly aggregate all the results in a very large group. Furthermore, we evaluate it using real-world Twitter and Facebook data. The data aggregation latencies for the two social networks show similar results, illustrating that `Spiral` can effectively process data from different social networks or platforms.

8) GROUP COORDINATION LATENCY

Group coordination latency refers to the latency of sharing a model among all groups. We use different models to evaluate the coordination latency by varying the number of groups (2, 4, 8, 16, 32, respectively). Note that the total number of agents in the system does not change, resulting in a relatively static communication hop between two groups' roots. We compare the coordination latency of the diffusion group and the star group. The star group refers to the peer-to-peer group that a root needs to communicate with, sending $n-1$ messages during one round time; in total, they have to send out $1/2 \times n \times (n-1)$ messages, therefore, taking $O(n^2)$ time. For the star group, as illustrated in Figure 9b, the group coordination latency increases rapidly as the number of groups and the model sizes increase. With the Random Forest (RF) models [36], the group coordination latency of 32 groups increases by about 700ms (3x) compared to the group coordination latency of 2 groups. When using Random Forest (RF) models (the size of RF model is near 10 Mb), the group coordination latency increases 10x than using small models (i.e. RT (Random Tree) [31], NB (Naive Bayes) [65], and Reg (Regression) [51] models are near 600 Kb). In comparison with the star group, the group coordination latency of the diffusion group is almost static (with a slight increase). This demonstrates that the diffusion group achieves low latency in the group coordination with $O(log(n))$ time complexity.

9) LATENCY OF PROCESSING CYCLES

Figure 9c shows the latency of different functions with two cycles of data processing in one group. It contains both local and global data processing with the model delivery time, data shuffling time, group communication time, and model delivery latency. Results show that the time of one processing cycle linearly increases as the number of agents increases. Differently from Figure 9b, where fixed-size groups achieve relatively static model delivery time, the increasing agents in trees result in a larger ring overlay in Figure 9c, and the average communication hops for the group roots increase. As a result, the model delivery time increases linearly, as illustrated in Figure 9c.

C. RUNTIME OVERHEAD

Spiral relies on a decentralized architecture (namely, the DHT overlay and hierarchical functional tree structure) that is able to distribute processing and overheads evenly over the agents. We evaluate the runtime resource consumption of a leaf agent in two processing cycles. As shown in Figure 10, CPU and memory utilization reach a relatively higher level from 5s to 30s and 35s to 55s, when the leaf agent is dealing with local data processing, which consumes more resources.

VI. CONCLUSION

In this paper, we present Spiral, a distributed system able to uncover social spam by leveraging the correlations between different social data sources in geo-distributed sites. Differently from prior work, which mostly focuses on offline spam detection, Spiral effectively handles the large-scale social data from geo-distributed sites in an online fashion. Through a novel approach that leverages DHT's consistent rings and publish/subscribe mechanisms, Spiral supports multiple groups to manage social data from various topics, areas, and geo-locations. Each group forms a functional tree that aggregates the properties of spam posts and creates updated spam classifiers to actively filter out the newest spam. Moreover, Spiral allows multiple groups to exchange and share spam correlations from distributed data sources, greatly enhancing the accuracy of spam detection results. Our real-work experiments demonstrate Spiral's scalability, effective load-balancing, and high-efficiency features for online and scalable spam detection.

An interesting question for future work is assessing the impact of inconsistencies inherent to distributed processing. Detecting spammers via the relational structure such as social connections and relationship graphs is a quite popular topic, so how to integrate the relational structure into Spiral? Existing work [42], [43], [70] discusses various approaches to use social graphs in spam detection, how can we scale the relational structures into a large-scale distributed environment? Furthermore, Spiral agents run in a highly dynamic environment with many unpredictable events such as network traffic interferences, workload interferences, software/hardware failures. As a result, some agents may run fast, while others may run relatively slowly. Some of the interesting problems worthy of further are: How to ensure the model consistency among distributed agents? How to scale the number of agents (e.g., scaling in/scaling out) and the size of Spiral groups based on the estimated peak rates of users' stream data and events? How can we protect the security and privacy concerns when processing data streams from different social media sites?

We will release Spiral as open-source, with all code and data used to produce the results in this paper.

ACKNOWLEDGMENT

The authors would like to thank the editors and the anonymous reviewers for their insightful suggestions and comments that improved this paper.

An earlier version of this paper was presented in part at the 12th International Conference on Cloud Computing [DOI: 10.1109/BigData.2018.8621926], [DOI: 10.1007/978-3-030-23502-4_11], and [DOI: 10.1109/CLOUD.2018.00020].

REFERENCES

[1] *Apache Flink*. Accessed: 2023. [Online]. Available: https://flink.apache.org/
[2] *Apache Flume*. Accessed: 2023. [Online]. Available: https://flume.apache.org/
[3] *Apache Giraph*. Accessed: 2023. [Online]. Available: https://giraph.apache.org/
[4] *Apache Hadoop*. Accessed: 2023. [Online]. Available: https://hadoop.apache.org/
[5] *Apache HBase*. Accessed: 2023. [Online]. Available: https://hbase.apache.org/
[6] *Apache Samza*. Accessed: 2023. [Online]. Available: http://samza.apache.org/
[7] *Apache Storm*. Accessed: 2023. [Online]. Available: http://storm.apache.org/
[8] *Kafka Ecosystem*. Accessed: 2023. [Online]. Available: https://engineering.linkedin.com/blog/2016/04/kafka-ecosystem-at-linkedin
[9] *MongoDB Atlas*. Accessed: 2023. [Online]. Available: https://www.mongodb.com/
[10] *Sha-1*. Accessed: 2023. [Online]. Available: https://en.wikipedia.org/wiki/SHA-1
[11] *Spark MLlib*. Accessed: 2023. [Online]. Available: https://spark.apache.org/mllib/
[12] (2016). *PHEME Dataset of Rumours and Non-Rumours*. [Online]. Available: https://figshare.com/articles/PHEMEdatasetofrumoursandnon-rumours/4010619
[13] (2019). *Graph API*. [Online]. Available: https://developers.facebook.com/docs/graph-api/
[14] (2019). *Twitter Standard Search API*. [Online]. Available: https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets
[15] (2019). *Twitter Usage Statistics*. [Online]. Available: https://www.internetlivestats.com/twitter-statistics/
[16] (2020). *How Does Twitter's Tweet Labeling Work?* [Online]. Available: https://www.dw.com/en/how-does-twitters-tweet-labeling-work/a-53622684
[17] (2021). *Twitter Rolls Out Redesigned Misinformation Warning Labels*. [Online]. Available: https://abcnews.go.com/Technology/wireStory/twitter-rolls-redesigned-misinformation-warning-labels-81212172
[18] (2021). *Twitter Tests New Labels for Misinformation, With Three Tiers of Alerts*. [Online]. Available: https://www.socialmediatoday.com/news/twitter-tests-new-labels-for-misinformation-with-three-tiers-of-alerts/601032/
[19] (2021). *What Facebook Knew About COVID-19 Misinformation and Didn't Tell Congress*. [Online]. Available: https://www.washingtonpost.com/politics/2021/10/28/what-facebook-knew-about-covid-19-misinformation-didnt-tell-congress/
[20] (2022). *5G*. [Online]. Available: https://en.wikipedia.org/wiki/5G
[21] (2022). *Blockchain*. [Online]. Available: https://en.wikipedia.org/wiki/Blockchain
[22] (2022). *Distributed Hash Table*. [Online]. Available: https://en.wikipedia.org/wiki/Distributedhashtable
[23] (2022). *Instagram Graph API*. [Online]. Available: https://developers.facebook.com/docs/instagram-api/
[24] (2022). *Linkedin API Overview*. [Online]. Available: https://learn.microsoft.com/en-us/linkedin/
[25] (2022). *Peer-to-Peer (P2P)*. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/peer-to-peer
[26] (2022). *Twitter API V2*. [Online]. Available: https://developer.twitter.com/en/docs/twitter-api
[27] (2022). *Twitter Transparency*. [Online]. Available: https://transparency.twitter.com/en.html
[28] (2022). *YouTube Data API Overview*. [Online]. Available: https://developers.google.com/youtube/v3/getting-startedi
[29] K. S. Adewole, T. Han, W. Wu, H. Song, and A. K. Sangaiah, "Twitter spam account detection based on clustering and classification methods," *J. Supercomput.*, vol. 76, no. 7, pp. 4802–4837, Jul. 2020.

[30] M. A. Al-Garadi, K. D. Varathan, S. D. Ravana, E. Ahmed, G. Mujtaba, M. U. S. Khan, and S. U. Khan, "Analysis of online social network connections for identification of influential users: Survey and open research issues," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–37, Jan. 2019.

[31] D. Aldous, "The continuum random tree III," *Ann. Probab.*, vol. 21, no. 1, pp. 248–289, Jan. 1993.

[32] Z. Alom, B. Carminati, and E. Ferrari, "A deep learning model for Twitter spam detection," *Online Social Netw. Media*, vol. 18, Jul. 2020, Art. no. 100079.

[33] M. J. Banda, R. Tekumalla, G. Wang, J. Yu, T. Liu, Y. Ding, E. Artemova, E. Tutubalina, and G. Chowell, "A large-scale COVID-19 Twitter chatter dataset for open scientific research—An international collaboration," *Epidemiologia*, vol. 2, no. 3, pp. 315–324, 2021.

[34] R. Böhme, N. Christin, B. Edelman, and T. Moore, "Bitcoin: Economics, technology, and governance," *J. Econ. Perspect.*, vol. 29, no. 2, pp. 213–238, 2015.

[35] A. Bovet and H. A. Makse, "Influence of fake news in Twitter during the 2016 U.S. presidential election," *Nature Commun.*, vol. 10, no. 1, pp. 1–14, Jan. 2019.

[36] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[37] J. Cao and C. Lai, "A bilingual multi-type spam detection model based on M-BERT," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–6.

[38] M. Castro, P. Druschel, A-M. Kermarrec, and A. I. T. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 8, pp. 1489–1499, Oct. 2002.

[39] J. Choi and C. Jeon, "Cost-based heterogeneous learning framework for real-time spam detection in social networks with expert decisions," *IEEE Access*, vol. 9, pp. 103573–103587, 2021.

[40] M. Cinelli, W. Quattrociocchi, A. Galeazzi, C. M. Valensise, E. Brugnoli, A. L. Schmidt, P. Zola, F. Zollo, and A. A. Scala, "The COVID-19 social media infodemic," *Sci. Rep.*, vol. 10, no. 1, pp. 1–10, 2020.

[41] M. Crawford, T. M. Khoshgoftaar, J. D. Prusa, A. N. Richter, and H. A. Najada, "Survey of review spam detection using machine learning techniques," *J. Big Data*, vol. 2, no. 1, pp. 1–24, Dec. 2015.

[42] P. De Meo, M. Levene, F. Messina, and A. Provetti, "A general centrality framework-based on node navigability," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 11, pp. 2088–2100, Nov. 2020.

[43] D. DeBarr and H. Wechsler, "Using social network analysis for spam detection," in *Proc. Int. Conf. Social Comput., Behav. Modeling, Predict.* Berlin, Germany: Springer, 2010, pp. 62–69.

[44] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Netw.*, vol. 14, no. 1, pp. 78–88, Jan./Feb. 2000.

[45] N. El-Mawass, P. Honeine, and L. Vercouter, "SimilCatch: Enhanced social spammers detection on Twitter using Markov random fields," *Inf. Process. Manage.*, vol. 57, no. 6, Nov. 2020, Art. no. 102317.

[46] E. Elakkiya, S. Selvakumar, and R. L. Velusamy, "TextSpamDetector: Textual content based deep learning framework for social spam detection using conjoint attention mechanism," *J. Ambient Intell. Hum. Comput.*, vol. 12, no. 10, pp. 9287–9302, Oct. 2021.

[47] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao, "Detecting and characterizing social spam campaigns," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, Oct. 2010, pp. 35–47.

[48] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2014, pp. 599–613.

[49] Y. Hassanzadeh-Nazarabadi, S. Taheri-Boshrooyeh, S. Otoum, S. Ucar, and Ö. Özkasap, "DHT-based communications survey: Architectures and use cases," 2021, *arXiv:2109.10787*.

[50] M. Himelein-Wachowiak, S. Giorgi, A. Devoto, M. Rahman, L. Ungar, H. A. Schwartz, D. H. Epstein, L. Leggio, and B. Curtis, "Bots and misinformation spread on social media: Implications for COVID-19," *J. Med. Internet Res.*, vol. 23, no. 5, 2021, Art. no. e26933.

[51] D. W. Hosmer Jr., S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, vol. 398. Hoboken, NJ, USA: Wiley, 2013.

[52] M. R. Islam, S. Liu, X. Wang, and G. Xu, "Deep learning for misinformation detection on online social networks: A survey and new perspectives," *Social Netw. Anal. Mining*, vol. 10, no. 1, pp. 1–20, Dec. 2020.

[53] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy K-nearest neighbor algorithm," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 4, pp. 580–585, Aug. 1985.

[54] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.

[55] F. Masood, G. Ammad, A. Almogren, A. Abbas, H. A. Khattak, I. U. Din, M. Guizani, and M. Zuair, "Spammer detection and fake user identification on social networks," *IEEE Access*, vol. 7, pp. 68140–68152, 2019.

[56] M. Mccord and M. Chuah, "Spam detection on Twitter using traditional classifiers," in *Proc. Int. Conf. Autonomic Trusted Comput.* Berlin, Germany: Springer, 2011, pp. 175–186.

[57] M. M. Najafabadi, "A research agenda for distributed hashtag spoiling: Tails of a survived trending hashtag," in *Proc. 18th Annu. Int. Conf. Digit. Government Res.*, 2017, pp. 21–29.

[58] M. E. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 64, no. 2, 2001, Art. no. 026118.

[59] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 40, no. 3, pp. 211–218, Jul. 2006.

[60] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The Bittorrent P2P file-sharing system: Measurements and analysis," in *Proc. Int. Workshop Peer-to-Peer Syst.* Berlin, Germany: Springer, 2005, pp. 205–216.

[61] S. Qaiser and R. Ali, "Text mining: Use of TF-IDF to examine the relevance of words to documents," *Int. J. Comput. Appl.*, vol. 181, no. 1, pp. 25–29, Jul. 2018.

[62] A. Rabkin and R. Katz, "Chukwa: A system for reliable large-scale log collection," in *Proc. 24th Large Installation Syst. Admin. Conf. (LISA)*, 2010, p. 163.

[63] K. K. Ramachandran and B. Sikdar, "A queuing model for evaluating the transfer latency of peer-to-peer systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 3, pp. 367–378, Mar. 2010.

[64] S. Rao, A. K. Verma, and T. Bhatia, "A review on social spam detection: Challenges, open issues, and future directions," *Expert Syst. Appl.*, vol. 186, Dec. 2021, Art. no. 115742.

[65] I. Rish, "An empirical study of the Naive Bayes classifier," in *Proc. IJCAI Workshop Empirical Methods AI*, 2001, pp. 41–46.

[66] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.* Berlin, Germany: Springer, 2001, pp. 329–350.

[67] D. Seyler, L. Li, and C. Zhai, "Semantic text analysis for detection of compromised accounts on social networks," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Dec. 2020, pp. 417–424.

[68] K. Shu, L. Cui, S. Wang, D. Lee, and H. Liu, "DEFEND: Explainable fake news detection," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 395–405.

[69] N. Spirin and J. Han, "Survey on web spam detection: Principles and algorithms," *ACM SIGKDD Explor. Newslett.*, vol. 13, no. 2, pp. 50–64, 2012.

[70] A. Srinivas and R. L. Velusamy, "Identification of influential nodes from social networks based on enhanced degree centrality measure," in *Proc. IEEE Int. Adv. Comput. Conf. (IACC)*, Jun. 2015, pp. 1179–1184.

[71] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, 2001.

[72] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna, "Understanding fraudulent activities in online ad exchanges," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf.*, 2011, pp. 279–294.

[73] Y. Sun and K. Loparo, "Opinion spam detection based on heterogeneous information network," in *Proc. IEEE 31st Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2019, pp. 1156–1163.

[74] S. Volkova and J. Y. Jang, "Misleading or falsification: Inferring deceptive strategies and types in online news and social media," in *Proc. Companion Web Conf. Web Conf.*, 2018, pp. 575–583.

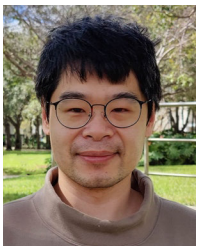
[75] B. Wang, A. Zubiaga, M. Liakata, and R. Procter, "Making the most of tweet-inherent features for social spam detection on Twitter," 2015, *arXiv:1503.07405*.

[76] L. Wu and H. Liu, "Tracing fake-news footprints: Characterizing social media messages by how they propagate," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, Feb. 2018, pp. 637–645.

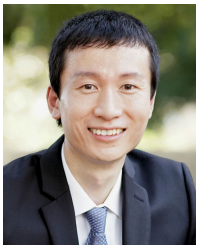
[77] L. Wu, F. Morstatter, K. M. Carley, and H. Liu, "Misinformation in social media: Definition, manipulation, and detection," *ACM SIGKDD Explor. Newslett.*, vol. 21, no. 2, pp. 80–90, Nov. 2019.
[78] Q. Wu, H. Zhang, X. Gao, P. He, P. Weng, H. Gao, and G. Chen, "Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems," in *Proc. World Wide Web Conf.*, May 2019, pp. 2091–2102.
[79] H. Xu, B. Guan, P. Liu, W. Escudero, and L. Hu, "Harnessing the nature of spam in scalable online social spam detection," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 3733–3736.
[80] H. Xu, L. Hu, P. Liu, and B. Guan, "Exploiting the spam correlations in scalable online social spam detection," in *Proc. Int. Conf. Cloud Comput.* Cham, Switzerland: Springer, 2019, pp. 146–160.
[81] H. Xu, L. Hu, P. Liu, Y. Xiao, W. Wang, J. Dayal, Q. Wang, and Y. Tang, "Oases: An online scalable spam detection system for social networks," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 98–105.
[82] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.
[83] A. Zareie and R. Sakellariou, "Minimizing the spread of misinformation in online social networks: A survey," *J. Netw. Comput. Appl.*, vol. 186, Jul. 2021, Art. no. 103094.
[84] Y. Zhang, R. Chen, and H. Chen, "Sub-millisecond stateful stream querying over fast-evolving linked data," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 614–630.
[85] Z. Zhang, R. Hou, and J. Yang, "Detection of social network spam based on improved extreme learning machine," *IEEE Access*, vol. 8, pp. 112003–112014, 2020.
[86] F. Zhou, X. Xu, G. Trajcevski, and K. Zhang, "A survey of information cascade analysis: Models, predictions, and recent advances," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–36, Mar. 2022.
[87] X. Zhou and R. Zafarani, "A survey of fake news: Fundamental theories, detection methods, and opportunities," *ACM Comput. Surv.*, vol. 53, no. 5, pp. 1–40, Sep. 2021.

HAILU XU received the B.S. degree in computer science from North China Electric Power University, China, in 2014, the M.S. degree in computer science from The University of Toledo, USA, in 2016, and the Ph.D. degree in computer science from Florida International University, in summer 2020. He is currently an Assistant Professor with the Department of Computer Engineering and Computer Science, California State University, Long Beach, USA. His current research interests include cloud computing, big data systems, and operating systems.

PINCHAO LIU received the B.Sc. degree (Hons.) from TJCU, China, the M.Sc. degree from TUST, China, and the Ph.D. degree in computer science from Florida International University, in 2021. He is currently a Research Scientist at Facebook (Meta) Inc. His current research interests include virtualization, cloud computing, and operating systems.

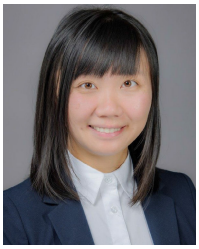


BOYUAN GUAN received the bachelor's degree from Beihang University, Beijing, China, the master's degree from Florida International University, Miami, USA, and the Ph.D. degree from the School of Computing and Information Science, Florida International University, in 2022. His research interests include operating systems, library systems, and big data.



QINGYANG WANG (Member, IEEE) received the B.Sc. degree in computer science and engineering from Wuhan University, in 2004, the M.Sc. degree in computer science and engineering from the Chinese Academy of Sciences, in 2007, and the Ph.D. degree from the College of Computing, Georgia Institute of Technology, in 2014. He is currently an Associate Professor with the Department of EECS, Louisiana State University, Baton Rouge. His research interests include distributed systems and cloud computing with a current focus on performance and scalability analysis of large-scale web applications (e.g., Amazon.com). He has collaborated extensively with scientists and researchers from industry companies, including IBM, HP, Amazon, Fujitsu, Japan, Wipro, India, and Huawei, China. He is also a Reviewer of IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *Journal of Parallel and Distributed Computing* (JPDC), and *ACM TOMPECS*. He is an Associate Editor of the *International Journal of Cloud Computing*.

DILMA DA SILVA (Senior Member, IEEE) received the Ph.D. degree in computer science from Georgia Tech, in 1997. She is currently a Professor and the Holder of the Ford Motor Company Design Professorship II at the Department of Computer Science and Engineering, Texas A&M University, USA. She is also an ACM Distinguished Scientist. Her research in operating systems addresses the need for scalable and customizable system software. She is a member of the Board of Computer Research Association's Committee on Widening the Participation in Computing (CRA-WP) and the Co-Founder of the Latinas in Computing Group.



LITING HU (Member, IEEE) received the bachelor's degree in computer science from the Huazhong University of Science and Technology, China, in 2007, and the Ph.D. degree in computer science from the Georgia Institute of Technology, USA, 2016. She is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California at Santa Cruz. Her research interests include distributed systems and its intersection with big data processing systems, resource management, power management, and system virtualization. She has served as an Editorial Board Member for *Future Generation Computer Systems* (FGCS) and the Program Co-Chair of the IEEE International Conference on Distributed Computing Systems (ICDCS).