This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



# Ring: Real-Time Emerging Anomaly Monitoring System over Text Streams

Weiren Yu, *Member, IEEE,* Jianxin Li, *Member, IEEE,* Md Zakirul Alam Bhuiyan, *Member, IEEE,* Richong Zhang, *Member, IEEE,* and Jinpeng Huai, *Member, IEEE,*

**Abstract**—Microblog platforms have been extremely popular in the big data era due to its real-time diffusion of information. It's important to know what anomalous events are trending on the social network and be able to monitor their evolution and find related anomalies. In this paper we demonstrate RING, a real-time emerging anomaly monitoring system over microblog text streams. RING integrates our efforts on both emerging anomaly monitoring research and system research. From the anomaly monitoring perspective, RING proposes a graph analytic approach such that (1) RING is able to detect emerging anomalies at an earlier stage compared to the existing methods, (2) RING is among the first to discover emerging anomalies correlations in a streaming fashion, (3) RING is able to monitor anomaly evolutions in real-time at different time scales from minutes to months. From the system research perspective, RING (1) optimizes time-ranged keyword query performance of a full-text search engine to improve the efficiency of monitoring anomaly evolution, (2) improves the dynamic graph processing performance of Spark and implements our graph stream model on it, As a result, RING is able to process big data to the entire Weibo or Twitter text stream with linear horizontal scalability. The system clearly presents its advantages over existing systems and methods from both the event monitoring perspective and the system perspective for the emerging event monitoring task.

**Index Terms**—Anomaly Detection, Real-time Anomaly Evolution Monitoring, Graph Model, Stream Processing

## 1 Introduction

MICROBLOG platforms have been extremely popular in the big data era due to the real-time nature and viral diffusion of information. A wide variety of anomalous events would emerge from such platforms, ranging from political or daily affairs to natural disasters or public security menace. These platforms have many times been the first reporter of significant events, such as earthquakes and accidents, or even the major hosting venue of significant events, such as a presidential election campaign. The value and application of a real-time emerging anomaly monitoring system over such platforms are many-fold. For instance, it can detect and respond to emergency events in a timely manner [1], [2], [3], [4], [5], track event evolution [6], [7], and summarize trending events [8], [9], [10].

The short and noisy nature of microblog text stream makes traditional topic detection methods and their derivatives inappropriate for the task [11], [12], [13], [14]. Latent space topic models is unsuitable for the fast changing online event monitoring scenario. The model takes a long time to train and the topics are fixed. While the online event monitoring scenario demonstrate fast changing set of topics and require high throughput of data processing. Nor are they able to perform early detection of emerging anomalous events in real-time. A topic model for short texts [15] has been developed, which has good performance on short texts. However, it is slower than LDA [13], hence cannot be applied in a real-time scenario. And later we will see that RING performs better than the method [15] in the event monitoring task. Systems scalable to full Twitter stream [2], [5], [16], [17] have also been proposed to detect trends, but they could not provide further correlation analysis into detected events. TwitterMonitor [16] provides online detection for general emerging anomalous events but could not reveal multiple aspects of the events nor track the evolution of them. CLEar [17] provides real-time anomalous event detection and tracking but could not provide correlation analysis of them. Neither of them could identify potential anomalous events before they are popular and spreading at scale. Signitrend [5] could detect potential anomalous events at small scale but is confined to only detection, and could not provide tracking and correlation analysis. The method would also tend to generate anomalous events that only contain a single keyword as description, which is hard to comprehend for users. None of the above methods provide the horizontal scalability with distributed implementations of their algorithms, nor do they investigate system optimizations for their applications. Cai et al. [18] has indexing system optimization combined with evolution tracking, but does not provide much of the detailed monitoring analysis RING provides, such as correlation analysis.

Despite the challenges, there are many *desired features* when building an emerging anomaly monitoring system over short text stream. From an *emerging anomaly monitoring* perspective, we require (1) early detection of emerging anomaly before they go viral, (2) hierarchical view of correlated sub-events as different aspects of an event, (3) monitoring and reveal the evolution of anomaly events,

*Manuscript received xxxx, 2016; revised xx, 2016; accepted xxx, 2016. Date of publication xxx, 2016; date of current version xxxx, 2016.*
*W. Yu, J. Li, R. Zhang and J. Huai are with the School of Computer Science and Engineering, Beihang University, Beijing, China. (e-mail: yuwr, lijx, zhangrc, huaijp@act.buaa.edu.cn).*
*M. Bhuiyan is with the Department of Computer and Information Sciences, Fordham University, New York, USA. (e-mail: mbhuiyan3@fordham.edu)*
*J. Li is corresponding author. W. Yu and J.Li contributed equally to this work and should be considered equal first authors.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



(3) generating highly correlated keyword summarization for better interpretation, (4) retrieving related representative tweets, (5) differentiating meanings of keywords under the effect of polysemy, (6) ranking events according to their importance and popularity, and (7) resilient to noise. Also from a *system* perspective, we need (1) stream processing to produce low latency outputs, (2) distributed processing to handle high throughput of data, and (3) scalability to manage peak data throughput.

Also we want to lift the two efficiency bottlenecks in our system, namely the full text indexing engine and the graph processing system, as they carry out heavy computation workload during anomaly detection and monitoring. Existing full-text indexing engines like TI [19] and Earlybird [20] do not take microblog's time feature into consideration, so they do not support the query with specific time range efficiently. There are several challenges for processing dynamic graphs in such scenario. Namely that existing systems [21], [22] take a lot of time during update of graph structure. They also cannot rebalance workload when some nodes in the graph are more frequently updated than others, which is a common case when processing popular words in microblog texts.

In this paper, we present RING, a real-time emerging anomaly monitoring system over microblog text streams. Emerging anomaly monitoring has attracted much attention from the research domain. Here we aim to monitor emerging anomalous events on microblog platforms. Our emerging anomaly monitoring methods are based on graph mining techniques, which provides unique opportunities to integrate our *emerging anomaly monitoring* research and *system* optimizations. In the RING system, emerging anomaly monitoring includes *early detection*, *correlation analysis* and *temporal evolution tracking* of anomalous events. *Early detection* would capture emerging events before they go viral. *Correlation analysis* would automatically reveal multiple aspects of the anomalous event, or the causality of anomalous events, or categorical structure of related anomalies. For example, the history and current development of a political event reveal multiple facades of the event. The capture of a criminal and the crime he had committed form a causal relationship. Different genres of news from an agency detected at the same time would reveal categorical information. According to its popularity, anomalous events would emerge from different time granularities, *e.g.*, a publicly concerned long trial or an overnight pop concert. The *temporal evolution tracking* of events could recover the evolution process of an anomalous event, to trace its origin and get the big picture. Such monitoring happens in real-time and provides valuable intelligence for government agencies, news groups and marketing agencies, etc.

To process big data, the design of RING graph model provides the following efficiency optimization opportunities. RING has a distributed graph processing engine specifically optimized for our anomaly detection method. Algorithms are implemented to have linear horizontal scalability to handle big data, *i.e.*, full stream of Weibo or Twitter data. The full-text indexing engine is optimized for efficient time range queries, which benefits our evolution tracking algorithms and queries over events and tweets. A user friendly interface is also provided to facilitate the analysis of emerging events with visualization.

Our major contributions in this paper include:

- We adopt anomaly detection method to monitor each keyword for early detection of trends. The proposed graph stream model is fully distributed with an efficient context statistics maintenance strategy and linear scalability.
- We provide a scalable anomaly monitoring approach meeting all the listed requirements. Especially, we are among the first to provide detailed correlation analysis of anomalies under the real-time emerging anomaly monitoring scenario.
- RING is among the first system to enjoy such rich set of anomaly monitoring features with dedicated system optimization efforts. The system optimizations of RING greatly improves the performance of emerging event monitoring.

We provide experimental evaluation to demonstrate the efficiency and effectiveness of our solution. The system is online for demonstration [23].

The paper is organized as follows: Section 2 introduces the graph stream model that combines our emerging anomaly monitoring research and system research. Section 3 introduces our methods for emerging anomaly monitoring. Section 4 gives the design and implementation of the underlying system and its optimizations. Section 5 evaluates our system in terms of anomaly monitoring efficiency and effectiveness and provides a comprehensive case study. Section 6 reviews related work. Section 7 concludes the paper.

## 2 GRAPH STREAM MODEL

We designed a graph stream model for anomalous event detection over the short and noisy text in microblog services. To represent texts with a graph stream model, we consider keywords as nodes and their co-occurrence relationships in each tweet as edges. For each incoming tweet, we generate a binary clique graph over its keywords and retrieve the edge set. That is to retrieve the edge of each keyword pair. The sequence of edges from the continuously arriving tweets generate the *graph stream data*. A table of symbols are summarized as in Table 1.

Table 1: List of Important Symbols in RING Graph Model

| Symbol | Description |
|---|---|
| $G(t)$ | Undirected temporal graph of keywords. |
| $\lambda$ | Decay rate of the weight of an edge arrival |
| $F(i,j,t)$ | The weight of an edge accumulated till current time $t$. |
| $S(i,t)$ | The set of neighboring nodes in node $i$'s locality |
| $\alpha(i,t)$ | The sum of the edge frequencies in locality set $S(i,t)$. |
| $HA(i,t,\lambda)$ | Half-life activity change of a node $i$. |
| $G_E(t)$ | Denoised keyword graph with rich information |
| $G_T(t)$ | Trending keyword graph which is binary |

An undirected temporal graph $G(t) = (N(t), A(t))$ is defined, whose nodes, edges and weight of edges change over time. We use $N(t)$ to denote the *set* of all distinct nodes in $G(t)$ at time $t$, and $A(t)$ as the *sequence* of edges received so far, respectively. The weight of an edge is defined as an accumulated frequency over all arrivals of the edges. To detect emerging events, greater importance is assigned to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



more recent arrivals of edges. Similar to [24], we apply a decay factor $\lambda$ to model the temporal effect, which regulates the decay rate of weight of an edge arrival, and to define the decay weight:

***Definition 1 (Decay Weight).*** At current time $t$, the weight of an edge arrival at time $t_s$ is defined as $2^{-\lambda \cdot (t-t_s)}$, with half-life of decay being $1/\lambda$.

Such smooth decay will avoid accidentally splitting a trend or peak. The sequence $A(t)$ would contain repetitions since we might receive an edge multiple times. We assume $A(t)$ contains edge $(i,j)$ in time $T(i,j,1)\dots T(i,j,n_{ij}^t)$, with a total of $n_{ij}^t$ times. In each timestamp $t$, edge $(i,j)$ is received $N(i,j,t)$ times. We define the weight of an edge at time $t$:

***Definition 2 (Weighted Frequency).*** The weight of edge $(i,j)$ at time $t$ is defined as the accumulated decay weights over all instances of its arrivals till time $t$.

$$F(i,j,t) = \sum_{k=1}^{n_{ij}^t} N(i,j,T(i,j,k)) \cdot 2^{-\lambda \cdot (t-T(i,j,k))} \quad (1)$$

Since the network is undirected, the value of $F(i,j,t)$ is the same as $F(j,i,t)$. We can see that the value of the frequency is often dominated by the relative recent arrivals of edges. We define the activity frequency of a node, the sudden change of which can be very useful to determine a trending keyword. At time $t$, we note the set of neighboring nodes of node $i$ as the locality set $S(i,t)$ and the index of the nodes in $S(i,t)$ as $\{j_1^i(t)\dots j_{|S(i,t)|}^i(t)\}$.

***Definition 3 (Node Activity Frequency).*** The node activity frequency $\alpha(i,t)$ of node $i$ at time $t$ is defined as the sum of the edge frequencies in its locality $S(i,t)$

$$\alpha(i,t) = \sum_{k=j_1^i(t)}^{j_{|S(i,t)|}^i(t)} F(i,k,t) \quad (2)$$

Let us now inspect the maintenance of these statistics under a streaming edge weight update scenario. We make the following observations with the definition of $F(i,j,t)$:

***Observation 1.*** As long as there are no new arrivals of the edge $(i,j)$ in time $(t',t)$, we have $F(i,j,t) = F(i,j,t') \cdot 2^{-\lambda \cdot (t-t')}$.

***Observation 2.*** When edge $(i,j)$ arrives, only the statistics of nodes $i$ and $j$ need to be updated.

We can see that updates to $F(\cdot)$ and $\alpha(\cdot)$ need to be explicitly performed only when new edges arrive. What's more, statistics of node $i$ and $j$ do not need to be explicitly updated when no edge is received of them. The update for each node $i$ could be independently processed in a distributed fashion as long as the node receives its own update data. During update, we first use Observation 1 to make statistics current till time $t$ from a decay perspective. The effect of new edge arrival is incorporated into $F(\cdot)$ by simply adding 1s. The computational complexity of this statistics maintenance is $O(|A(t)|)$, which is proportional to the edge update received.

We now have a temporal keyword co-occurrence graph. The weighted frequency of edges measure the *temporal association* between two keywords and the activity frequency of nodes measure the *activity level* of a keyword. The context of a keyword is well preserved in its locality $S(i,t)$. For each node $i$, three pieces of information are maintained. Specifically, they are (i) the last time stamp $L(i)$ at which an edge was received of node $i$ (ii) the set of nodes in $S(i,t)$ (iii) an array of frequency values $F(i,j,L(i))$ for each node $j \in S(i,t)$. Note that the space requirement of this maintenance is proportional to the sum of the degrees of the nodes in the graph. A keyword graph is relatively sparse hence the value of $|S(i,t)|$ is typically much smaller than the number of nodes in the network, which makes the maintenance compact and efficient in the streaming scenario.

## 3 Emerging Anomalous Event Monitoring

In this section, we introduce RING's approach to the emerging anomalous event monitoring. The process is as illustrated in Figure 1. Emerging anomalous event monitoring is defined as event detection, correlation analysis and temporal evolution tracking of events. *Early detection* could detect currently popular events. And it also aims to discover potential viral events at an early stage. *Correlation analysis* reveals the relationship among events and provides insight for better analysis of these events. *Temporal evolution tracking* traces the evolution pattern for events of different lifecycles and popularity, *e.g.,* from hours to months. An event refinement process performs spam filtering before and after texts are distilled into events. It also performs context enrichment for events to facilitate the tracking of them, such as entity recognition, geo-location identification etc.

### 3.1 Early Detection

The anomaly detection method consists of two steps: trending keyword detection and community detection over keywords. Our intuition is that the essential keywords about an event would have similar trends and show a burst in usage compared to their own history. Trending keywords are detected as an anomaly with abrupt usage increase. An event is represented as a "bag" of keywords that co-occur and correlate with each other, with its detection time and representative tweet extracted for better comprehension. We shall see that trending keywords and their co-occurrence relationship is a strong combination to distill emerging anomalous events. These signals would also reveal hierarchical event correlation, which will be introduced in the next section (Section 3.2). Valuable intelligence, such as different aspects of events, causality of events or categorical structure of events, could be revealed through the correlation analysis.

#### 3.1.1 Trending Keyword Detection.

We consider trending keyword detection task as an outlier detection for bursty usage. We define the activity level change for a node. Typically, such change is most informative when the time period is $1/\lambda$, $i.e.$ the half-life of edge weight decay.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



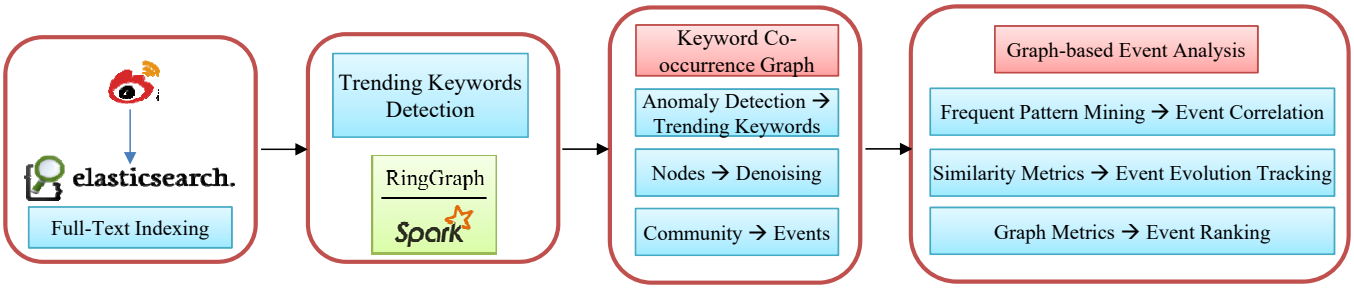



Figure 1: RING Emerging Event Monitoring Procedure

***Definition 4 (Half-life Activity Change).*** The half-life activity change $HA(i,t,\lambda)$ of node $i$ at time $t$ is defined as $\alpha(i,t) - \alpha(i,t-1/\lambda)$.

By measuring $HA(i,t,\lambda)$ over each half-life period, we would get a series of $HA(i,t,\lambda)$ values, from which a good estimate of the magnitude of activity change could be obtained. It essentially measures the acceleration of each keyword's usage hence unusual bursty activity of keywords could be detected before it would go viral, regardless of its traffic. We then apply the standard score method to determine the unusual deviations in activity change. According to quantification from Gao et al. [25], this could serve as a good estimate for our purpose. The standard score, or *outlierness score*, can be computed as follows:

$$ZValue = \frac{HA(i,t,\lambda) - \mu^A(i,t,\lambda)}{\sigma^A(i,t,\lambda)} \tag{3}$$

where $\mu^A(i,t,\lambda)$ and $\sigma^A(i,t,\lambda)$ are the mean and standard deviation of the series. When the value of this deviation is larger than a threshold $ZThresh$, the node would be flagged as a trending keyword. Empirically we set the value of $ZThresh$ to 3, which corresponds to 99.7% of normal data. Increase of $ZThresh$ will yield less trending keywords but stronger burstiness. The cost of the detection operation on each half-life period is linear to the sum of degrees of the updated nodes. The number, sum and square sum of $HA(i,t,\lambda)$ values are maintained for each node in the graph for incremental update in a streaming fashion.

*Multi-granularity Analysis.* Keywords of an emerging event would demonstrate similar trends within the same time granularity. But the period over which trends would emerge is often unknown in advance. Over longer periods of time edge frequencies will stabilize while over shorter periods it will be sensitive to changes. Parameter $\lambda$ controls the level of temporal granularity for analysis. For trending keyword detection, we can choose a few approximate ranges and monitor each $\lambda$ independently. Empirical trend periods would vary from 10 minutes to a few hours in microblog services. For larger values of $1/\lambda$, the monitoring for $HA(i,t,\lambda)$ series is less frequent due to extended monitoring periods and requires less computation.

*Keyword Denoising.* We now have a keyword graph consisting of the trending keywords, their co-occurred keywords and their statistics. At each detection period, we perform noise filtering over both kinds of keywords. We first try to remove unrelated co-occurred keywords that have not appeared with a trending keyword recently. To do this, we delete edges in a trending keyword's locality with less weighted frequency than a threshold *edge_freq_min*. By default, we set it to 1. We further recursively remove nodes with unweighed degree not larger than 1. Irrelevant co-occurred keywords that only co-occur with one trending keyword are hence effectively removed. Such keywords would belong to some other contexts of the trending keywords. A "standalone" or "single-handed" trending keyword is considered insignificant or it would have necessarily co-occurred with other trending keywords. Such trending keywords are often revealed after the irrelevant co-occurred keywords are removed. Results show that the above procedures would usually remove at least half of the nodes in the keyword co-occurrence graph.

Our method is naturally resilient to noisy keywords absent of bursty features, such as words about trivial things of mood, food etc., since these usage are relatively stable in the big picture. We note that the more the data, the better the chance to acquire accurate keywords with similar trends, the more meaningful the results would be.

#### 3.1.2 Event Extraction.

Keywords of the same event would co-occur and be more densely linked internally than with keywords from a different event. It would be intuitive to follow modularity-based [26] or betweenness centrality based [27] community detection methods for event detection. However, such methods would lead to 4 major drawbacks: (1) non-local: density changes in parts of the graph would affect the overall result for community detection. (2) mixed contexts: a keyword cannot appear in different events (3) co-occurrence by chance: keywords in different contexts would likely to be assigned to the same event due to lack of explicit definition of keyword community (4) no hierarchy: hierarchical correlation analysis is absent for such methods. Events would naturally consist of different aspects which formulate a hierarchical structure. Next we would introduce our method for event detection and hierarchical sub-event correlation analysis.

We now have a denoised weighted keyword graph $G_E(t)$ which maintains statistics of trending keywords, namely the detected period, outlierness score, co-occurred keywords and co-occurrence frequency.

***Definition 5 (Denoised Keyword Graph).*** Denoised keyword graph is a directed weighted graph $G_E(t) =$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



$(V_E(t), E_E(t))$, $V_E(t)$ contains all keywords after denoising, while $E_E(t)$ contain edges among all keywords with their weighted frequency. The direction of an edge starts from a trending keyword and points to an arbitrary node.

We also define a binary graph of only trending keywords:

***Definition 6 (Trending Keyword Graph).*** Trending keyword graph is a binary graph $G_T(t) = (V_T(t), E_T(t))$, where $V_T(t)$ contains only trending keywords and $E_T(t)$ are edges between trending keywords.

$G_T(t)$ extracts the co-occurrence pattern between trending keywords from $G_E(t)$ as edge structures. Event detection is performed on $G_T(t)$ while $G_E(t)$ provides additional information for event denoising, ranking and extension with co-occurred keywords. By definition, two keywords would not have appeared in the same text if there were no edge between them. In other words, keywords form the finest granularity of event aspects in their *1-hop* neighborhood. Then the *maximal cliques* in the graph correspond to explicit definition for aspects of events with strong correlation, where all keywords co-occur with each other.

To distill events from these aspects and get the bigger picture, we use *k-clique percolation* to find overlapping communities as events [28]. A *k-clique-community* is defined as a union of all k-cliques that can be reached from each other through a series of adjacent k-cliques, where adjacency means sharing *k-1* nodes. Such definition of community enjoys several advantages versus existing methods: (1) local definition of community that would not be affected by change in parts of the graph (2) natural overlap to decode polysemy and diverse contexts (3) reduce co-occurrence by chance through parameter $k$ (4) hierarchical view of sub-events is encoded in the maximal cliques of the communities.

The *k-clique percolation* method finds all maximal cliques and test maximal clique overlaps to extract percolated communities. Parameter $k$ directly controls the minimal number of trending keywords a clique should have. As such, any event with less than $k$ trending keywords would be discarded. A larger $k$ value would implement a stronger correlation constraint on forming a community, as it gets harder to form a $k$-clique by chance. We set $k$ to 3 by default in order to get more events. Empirical evaluation show that such setting would yield relatively high quality results while preserving more aspects of events than larger $k$.

For a weighted graph, there would be another parameter for the method, namely edge frequency threshold $w^*$. We note that $w^*$ is equivalent to *edge_freq_min*. We do not set larger values of *edge_freq_min* at this stage to preserve details, *i.e.*, event aspects that's represented by low weight edges. We define the event and its aspects over $G_T(t)$ as follows:

***Definition 7 (Trending Event).*** A trending event is defined as a k-clique-community of binary graph $G_T(t)$, where k is 3 by default. The maximal cliques under each community constitutes aspects of events.

**Algorithm 1** Sub-event Hierarchy Construction

**Require:** $G_T^{'}(t)$ and its maximal cliques, parameter $k$
**Ensure:** Branches of FP-tree as hierarchical sub-events
1: Put nodes in each maximal clique in descending order according to their degree in $G_T(t)$.
2: Build a frequent pattern tree with each maximal clique
3: Check under depth of k.
4: **if** the number of nodes two sub-trees share $> k-1$ **then**
5: sub-trees belong to the same sub-event
6: **else**
7: sub-trees are different sub-events.
8: **end if**

*Scalability.* The size of $G_T(t)$ is guaranteed by the three sigma rules to be approximately 0.3% of the graph size, which is a modest scale. We apply an optimized method [29] of k-clique percolation which minimizes clique overlap test for efficiency. Further, by first computing the disconnected subgraphs of $G_T(t)$, which is a linear operation, parallelization could then be achieved. As would be shown in experiment, these methods are efficient to achieve real-time performance.

*Event Denoising.* For each event we retrieve its maximal edge frequency from $G_E(t)$. To filter out noisy trending events with low frequency, we pick out communities with maximal edge frequency less than *event_freq_min*, The threshold is set to 2 considering the volume of data we have and to preserve hints of trends. After event denoising, we note *k-clique-communities* in $G_T(t)$ as $G_T^{'}(t)$.

### 3.2 Correlation Analysis

We can detect emerging events of strong keyword correlation with the above method. However, at a finer granularity, we observe that different aspects of an event would emerge in a single event. Since the events are extracted from clique adjacency relationship, there would exist maximal cliques in a community that are not directly adjacent to each other. This indicates potential different aspects of an event. These aspects may entail categorical information, causality relationships or different opinions on the same event. Sometimes these aspects could just co-occur by chance.

To reduce co-occurrence by chance, summarize event aspects into sub-events and build a hierarchical view of the event, we devise Algorithm 1, an algorithm based on frequent pattern tree (FP-Tree) [30], which assigns cliques sharing $k$ or more nodes as the same sub-event and separates maximal cliques sharing *k-1* nodes or less. Cliques that are not directly adjacent to each other but belong to the same *k-clique-community* are assigned to different sub-events, differentiating different aspects of the event. A modularity based or betweenness centrality based clustering method would likely to mix different sub-events together. Valuable intelligence, such as different aspects of events, causality of events or categorical structure of events, could be revealed. We are the first to provide a hierarchical view of correlated events for the emerging event detection scenario. Event related tweets are retrieved as representation texts through querying the search engine with keywords of events.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



100 detected real world events are manually inspected and its most related tweet in our dataset. We find that the method can detect 47% of the emerging events within 5 minutes and 79% of events within 10 minutes after the first tweet appeared in the event. The method can outperform state-of-the-art topic and event detection methods based on keyword co-occurrence [15], [26], [27] in terms of keyword coherence (NPMI) and summarization quality (ROUGE-1). The method is implemented on Spark and share a cluster of 8 machines with other algorithms and system components, where it can handle 16k tweets per second with linear horizontal scalability.

### 3.3 Temporal evolution tracking

Event tracking aims to trace temporal evolution of events along the timeline. Given a latest event $E_0$, the evolution chain of $E_0$ is defined as follows:

$$E_1 \rightarrow E_2 \rightarrow \ldots \rightarrow E_n \rightarrow E_0 \tag{4}$$

where $E_i$ is a precedent event of $E_0$ and $E_i \rightarrow E_j$ means event $E_j$ developed from $E_i$. Under the assumption that related events share at least one noun in their keyword sets, a candidate set $CS$ of potential precedent events is retrieved using nouns in the keyword set of $E_0$. An inverted index is built to map nouns to events. In practice the full-text search engine is utilized to provide query results. The event chain of $E_0$ is built through a set of similarity metrics of the location $loc_i$, participant set $ps_i$, and event related post set $ws_i$ of event $e_i$ in $CS$. The similarity of two events $e_i$ and $e_j$ are defined as follows:

$$\begin{aligned} Sim(e_i,e_j) = \alpha \cdot Sim_{ws}(ws_i, ws_j) + \\ \beta \cdot Sim_{loc}(loc_i, loc_j) + \gamma \cdot Sim_{ps}(ps_i, ps_j) \end{aligned} \tag{5}$$

$Sim_{ws}(ws_i, ws_j)$ is based on average cosine similarity of related posts. $Sim_{loc}(loc_i, loc_j)$ equals 1 if $e_i$ and $e_j$ share the same location and 0 otherwise. $Sim_{ps}(ps_i, ps_j)$ is based on Jaccard similarity of participant sets $ps_i$ and $ps_j$. $\alpha + \beta + \gamma = 1$ and can be adjusted empirically. $Sim(e_i, E_0)$ is calculated for each event in $CS$ and note set $Chain$ for $E_0$: $Chain = \{e_i \in CS | Sim(e_i, E_0) > \varepsilon\}$. Threshold $\varepsilon$ is used to filter out potential unrelated event candidates. It is worth mentioning that to merge near duplicate events in $Chain$ a separate threshold $\varepsilon'$ is devised. $\varepsilon'$ is larger than $\varepsilon$ and smaller than 1.

Our method [31] leverages locality sensitive hashing to find similar events efficiently. What's more, time ranged query in the underlying search engine system is specifically optimized to improve its efficiency. The time range controls how far back we would like to trace. And with system and algorithmic optimization, our method is effective in revealing the evolution of both the long-term and short-term events, rather than only recent events [32], [33].

### 3.4 Event Refinement.

The refinement procedures are responsible for spam filtering and context enrichment. Based on [34], the spam accounts are first removed from our data crawler, who are either constantly publishing ads content or actually manipulated by intelligent software for propaganda purposes. A 3-class

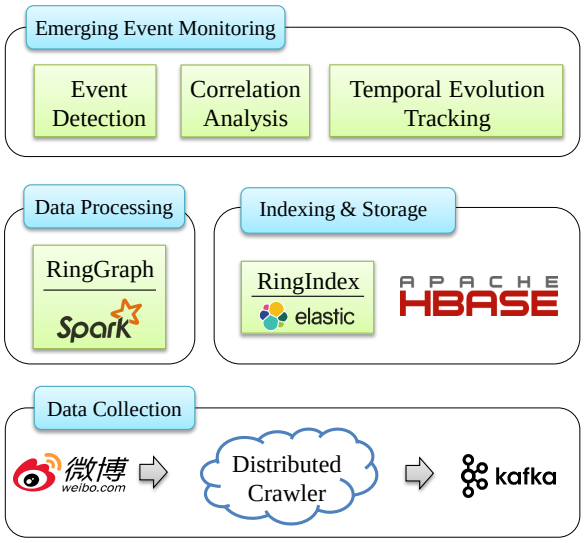



Figure 2: The Architecture of RING System

Naive Bayes is applied to differentiate news, ads and widsom words among detected trending events, where the latter two classes are major types of spams on Weibo. The classifier is trained with manually labeled data based on features of content, users and temporal information. Further the location of the detected events are extracted using a location vocabulary and the nouns and candidates of events with *ICTCLAS*[1].

For keywords and events, importance and popularity comes naturally from the burstiness score and edge frequency. A larger burstiness score would indicate the keywords and events are becoming popular quickly. A larger edge frequency would indicate keywords and events have significant amount of traffic and may already be popular. In practice, it is comprehensive to rank keywords according to weighted degree and events with highest edge frequency.

We use trending keywords in each maximal clique of the event as query and retrieve perfect matching tweets from their detected period as summarization. The co-occurrence relationship in a clique indicates a high probability that a matching tweet exists. There are cases of mismatch for a very large $k$ (*e.g.*, 15). Only subsets of the $k$ keywords would appear in a single tweet as a result of synonyms. Hence it gets harder to find a tweet containing all $k$ keywords. It is also possible that the mismatch comes from accumulated decay effects of edge occurrence in previous time periods.

## 4 THE RING SYSTEM

As shown in Figure 3, RING system mainly consists of three modules, namely *data collection*, *data indexing & storage*, and *data processing*. For *data collection*, a distributed crawler is developed to continuously fetch microblogs through Weibo API[2]. The collected data is forwarded to indexing and processing modules through Kafka[3] to decouple their dependency. For *indexing and storage*, RING utilizes HBase[4]

1. http://ictclas.nlpir.org/
2. http://open.weibo.com/wiki/Api
3. http://kafka.apache.org
4. http://hbase.apache.org/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



and elasticsearch[5] for data storage and full-text indexing, which are able to process large volumes of real-time microblog streams. Time range query of elasticsearch is further optimized for event tracking and keyword queries. For *data processing*, our models and algorithms are implemented on Spark[6]. An optimization strategy for incremental graph statistics update is also devised to improve distributed processing efficiency.

### 4.1 Data Collection

A distributed crawler is built to collect data from Weibo, the largest microblog platform in China. The crawler continuously collects the latest microblogs published by users preferrably with a large number of followers, *i.e.*, opinion leaders. The crawler has a master/slave architecture. The master node utilizes key/value store to perform task scheduling. Slave nodes get the assignments from the master and crawl data. A task would monitor the reposts and comments of an original tweet and retrieve the repost and comment list, from which we can construct the forward graph of each tweet. The tasks are scheduled according to posts' priority, which is weighed by the number of reposts and comments. Each task has a life cycle so that the monitoring for each post can be effectively maintained or terminated. The data is crawled using Weibo API. Up to the time of writing we are able to collect 3 - 10 million microblogs daily.

### 4.2 Indexing Engine

For microblog platforms, real-time search is frequent. It's for the reason that most users are interested in microblogs posted in recent time. More importantly, history search is vital to event evolution tracking application. The application issues extensive amount of keyword queries to retrieve related tweets in the text archive for similarity computation. Each query is issued for a specific time interval as defined in the event detection algorithm. Both real-time search and history search can be treated as the search with a specific time range, with various time range for query.

Up to now, there have been several studies to solve realtime search problem on microblog system. Earlybird [20] , a real-time search engine for twitter, builds efficient in-memory indices and can execute a query with an average latency of 50 ms and make new tweets searchable within 10 seconds. TI [19] classifies the tweets by query logs and only index the distinguished tweets in real-time in order to decrease index update cost. However these works do not take microblogs time feature into consideration, so they do not support the query with specific time range well.

To overcome the limitation of existing methods, we propose RINGINDEX, a distributed online index system for temporal microblog data. The whole index is divided into fine-grained time range partitions to provide locality for data access according to temporal approximity. For better concurrent access, each time range partition is divided into sub-partitions by hash functions. Each term's inverted tweet list is only mapped to corresponding sub-partitions. With these structures, given a query with a specific time range,

5. https://github.com/elastic/elasticsearch
6. http://spark.apache.org

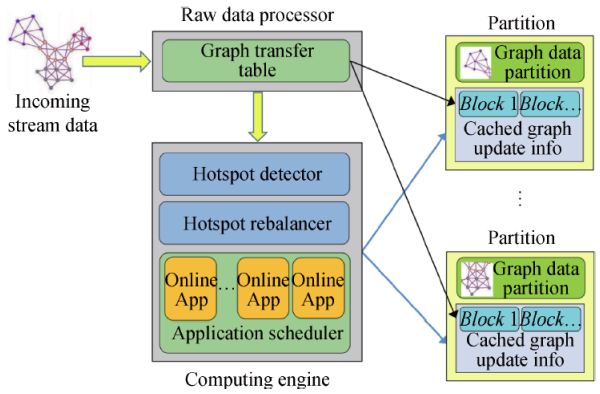



Figure 3: RING Graph Processing System

the time range information can be used to navigate to corresponding time range partitions and then utilize query to quickly navigate to corresponding sub-partitions.

In addition, an index chain is also adopted to merge terms with the same post list to reduce the size of index. The optimization technique is distributed, each node having the same index structure. Whether updating index or searching can be carried out on each node at the same time, which can help provide better concurrent processing ability and further reduce the index and query cost. The optimization technique is universal and implemented in elasticsearch. Time range structure is implemented as Time Range Partition Index Set (abbrv. TRPIS) in the multi-layer index of elasticsearch to directly optimize time range query performance [35]. Such structure for time ranged queries avoids traversing each layer and allows efficient navigation to records within a specified time interval.

### 4.3 Graph Processing Optimization

As discussed in Section 3.1, a continuously changing graph of keywords and their co-occurrence relationship is built from the text stream. To improve the efficiency of graph update and computation process, an optimized system, RINGGRAPH is developed based on Spark and GraphX and implemented our event monitoring system on it.

There are several challenges for processing dynamic graph in such a scenario. The first is that modifications of the graph structure tend to be time-consuming in existing systems, especially for inserting new nodes and new edges [36]. Since for the RING event monitoring algorithm, nodes in the keyword graph will continue to increase as time passes, we need an efficient graph structure update mechanism. The second is that real-time processing is not easy to be achieved, because most of the existing graph-parallel systems [37], [38], [39], [40], [41], [42], [43], [44], [45] process graph in a global and batch mode, rather than incremental mode. Along with the huge scale of social graphs, the time cost of each iteration is unacceptable [46]. The third is that the time-sensitive feature of dynamic graph may lead to workload imbalance, because some of the graph nodes are much more frequently updated than others in a particular period of time.

In the RING event monitoring, the updates of trending keywords, whose usage would have a spike in a short

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



period of time, will show uneven update behavior. To efficiently update graph structure, we introduced a hash-based graph partitioning method to support find-grained and rapid update. To support incremental computation on evolving graphs, we designed an application scheduler to coordinate applications with different computing request frequency and control the use of compute resources. Specifically, the scheduler is in charge of merging the graph update data into the main graph, triggering computing task for applications and releasing resources of the cached graph update information. In order to support incremental algorithms, we introduce a vertex-based graph computation model, which is highly fit to most of the scenarios.

To address the workload imbalance issue, a workload hotspot detection and balancer is designed to deploy computing tasks according to data locality and compute resources while minimizing the migration cost. What's more, a hash-based graph partitioning method [47] is adopted to distribute computation. To make our system support applications that simultaneously execute computing requests on the graph with different frequencies, an application scheduler is implemented to efficiently exploit the resources. The application scheduler consists of two components: the hotspot detector, which adopts a standard score based graph stream anomaly detection algirithm [24] to identify compute nodes and partitions with heavy workloads in real time, and the hotspot rebalancer, which uses different task distribution algorithms to migrate compute tasks. The distribution algorithms optimize data localization or minimize data transfer during migration.

## 5 Experiments

In this section, the RING system is evaluated in terms of system efficiency and event monitoring quality.

### 5.1 Overall Settings

For RING system evaluation, we use the following dataset and testbed for all experiments.

**Dataset:** A total of 152,964,368 tweets are used crawled from Sep.11.2014 to Oct.27.2014 on Weibo. IKAnalyser is used for tokenization and WebDict as vocabulary. The vocabulary contains 201,197 words. Stop words are removed and tweets that contains less than 5 words are filtered out. We analyzed cases from the production RING system to demonstrate event evolution monitoring in Section 5.3. We also used social graph data in Section 5.6.

**Testbed:** We used a shared cluster of 8 machines, each with 2 Xeon E5-2650 CPUs (16 physical cores) and 256GB RAM connected through Infiniband @40Gb/s. The machines run Debian7 and JDK 1.7, both in 64-bit.

**Tunable Parameters:** We list tunable parameters in Table 2.We would evaluate the effects of $1/\lambda$ and $k$, which are the most semantically effective parameters, and leave other parameters fixed to their default values.

### 5.2 Effectiveness of Anomaly Detection

We evaluate the effectiveness of our approach through various metrics and inspect the effects of $1/\lambda$ and $k$ . Unless specified, we use default values of $1/\lambda$ and $k$ as shown in Table 2. Chinese words are translated to English.

Table 2: Tunable Parameters

| Parameter | Default | Description |
|---|---|---|
| $1/\lambda$ | 10′ | half-life of decay |
| $k$ | 3 | k-clique percolation |
| ZThresh | 3 | burstiness thresh |
| $edge\_freq\_min$ | 1 | minimum edge freq |
| $event\_freq\_min$ | 2 | event frequency thresh |

**Methods For Comparison.** We note our approach as RING and include 3 other methods for comparison. Using modularity based method [48] for event detection was introduced by Weng et al. [26]. We implemented the clustering method [48] and replaced RING's event detection method for comparison, *i.e.,* perform modularity based clustering [48] on $G_E(t)$. We note this approach as RING-M. We set resolution of this method to 1.0 by default. We included two topic models based on co-occurrence relationship for comparison, namely graphical model Biterm [15] and graph model KeyGraph [27]. We used published codes of the authors to run the experiment. For KeyGraph we used its default short text parameter setting. For Biterm we set number of topics $K$ to 20 for each time slice. Data was partitioned into slices of each half-life period. Biterm and KeyGraph methods were run on each data slice to get comparative results. Since the method for comparison perform similar tasks, the default parameters also correspond to their best reported performance in the original paper. We verified that this is also true on our dataset, by running our quantative evaluation methods and by manual evaluation where the results do not contain many duplications and confusing results.

#### 5.2.1 Early Detection Analysis

In general, RING would detect more than 200 events in a day, and most of them would be picked up by news media. To demonstrate the sensitivity of our method to potential trends, we manually picked 100 events out of the detected trending events with half-life being 10 minutes. The 100 events are real world events picked on 8 random days from Sep.11.2014 to Oct.10.2014. These events were all viral real world events on the Weibo platform that made to the top popular event charts, and were covered by multiple news media, include blogs, news websites etc. We calculate the response time of our method as $T_1$-$T_2$, where $T_1$ is the detection time and $T_2$ is the time of its first tweet. The first tweets are identified manually using ElasticSearch[7] .

**Result Analysis.** From Table 3 we can see that 79% of the events can be detected within 10 minutes, with 47% detected within 5 minutes. The average response time over these 100 events is 18 minutes. The detected events do not always have a large traffic. Accelerated spreading would be verified if it is followed by immediate subsequent detections. This would mean that the anomaly event had successive exponential growth and was detected in consecutive periods. We can see that our method is sensitive to trending events and can detect them at an early stage. We discover that events

7. http://elaticsearch.org

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



Table 3: Response Time in Minutes

| Response Time | Count | Sum of Percentage |
|---|---|---|
| 0-5 | 47 | 47% |
| 6-10 | 32 | 79% |
| 11-15 | 5 | 84% |
| 16-20 | 2 | 86% |
| 21-65 | 7 | 93% |
| >65 | 7 | 100% |

with over 30 minutes of response time lack burstiness in the pre-detection period.

#### 5.2.2 Effects of Multi-Scale Anomaly Detection

We used half-life period of 10, 30 and 60 minutes for trending keyword detection.

**Result Analysis.** There are on average **2188**, **1730** and **938** trending keywords respectively for half-life period of 10, 30 and 60 minutes. As the half-life extends, there are less detected trending keywords and less detected events. We observe that detected trends over 10-minute periods reveal subtle changes in the main point of the event. Trends observed under longer half-life period usually correspond to significant events that have a consecutive exponential growth in volume. The system should "ring" the alarm when such trends are detected.

#### 5.2.3 Keywords Coherence

Pointwise mutual information (PMI) [49] is a good evaluation metric for keywords coherence evaluation, which has been proven to have near inter-annotator correlation performance and preferred by researchers [50]. We used a normalized version of the metric to assign its value range to [-1,+1]. This avoids negative infinity values and neutralizes its preference for low frequency events [51]. The definition is as follows:

$$NPMI(x,y) = -1 \cdot \frac{1}{log[p(x,y)]} \cdot log\frac{p(x,y)}{p(x)p(y)} \quad (6)$$

where $x$ is a random variable and $p(x)$ is defined as the frequency of $x$ in the document collection. The final NPMI score of an event E is defined as the mean of all $NPMI(w_i, w_j)$ values of its $K$ keywords:

$$NPMI(E) = \frac{2}{K(K-1)} \cdot \sum_{1<i<j<K} NPMI(w_i, w_j) \quad (7)$$

We calculate the NPMI score for an event against data in the corresponding time slice. Out of the 100 emerging events discovered by RING, we manually selected 17 events that are also detected by all other methods. The result of mean and standard deviation of the scores are shown in Table 4. An intuitive and detailed comparison of the scores is presented in a case study.

**Ring Analysis.** We can see that RING outperforms other methods significantly. The coherence score advantages come from 4 factors: (1) trending keywords found by anomaly detection method have similar trends hence coherent with each other. (2) co-occurrence by chance is minimized by coherence constraint of maximal clique structures. (3) clique percolation tends to find more aspects of the event with strong correlation. (4) keywords are allowed to overlap between events, yielding a more natural result.

Table 4: Normalized PMI Results

| Method | | Median | Mean | $\sigma$ |
|---|---|---|---|---|
| KeyGraph | | 0.3841 | 0.4690 | 0.2336 |
| Biterm | | 0.6458 | 0.5964 | 0.1943 |
| RING-M | | 0.6528 | 0.6249 | 0.1529 |
| RING | $k$=3 | **0.7546** | **0.7458** | **0.1468** |
| | $k$=4 | 0.7609 | 0.7780 | 0.1470 |

**Comparison Analysis.** KeyGraph applies betweenness centrality based clustering which tends to include co-occurred keywords by chance. Such unrelated words would hurt coherence. So does modularity based method since they both lack a precise definition for community. Result for Biterm is consistent with its own reported performance [15]. Biterm directly models the generation process of word co-occurrence patterns, which yields a good performance. But it could miss events with small magnitude absent of trending keyword detection phase and lacks a mechanism to explicitly filter out unrelated co-occurrence patterns.

**Parameter Sensitivity.** In Definition 7, we define event of trending keywords as a *k-clique community* where $k = 3$. Increasing $k$ would implement a stronger semantic coherence constraint on keywords of the detected event. We see that increasing $k$ would improve NPMI scores as expected. Note that increasing $k$ would unlikely lead to discarding the detected events, only a re-selection of the keywords.

**Case Study.** In Table 5 we demonstrate an event detected by four methods. RING is able to distill 4 sub-events of this game, which are different aspects of the event. This shows that RING can detect and differentiate events at a resolution other methods cannot. The case is about the semifinal game at ATP Masters 1000 Shanghai where Roger Federer beat Djokovic and got into the final. Djokovic has had 28 straight wins in China and Federer will face Gilles Simon in the final. We see that the first two sub-events seem to be the same. This shows that Algorithm 1 could sometimes produce duplicate information. The third sub-event is about the straight win of Djokovic and the fourth is about the result of the semifinal game. Again we note that in RING system a representation tweet is provided to help users understand what is happening.

Let us look at the NPMI scores in Table 5. Keywords of each sub-event in RING are highly coherent. KeyGraph got the highest score but only detected 3 keywords for the event. Biterm got the lowest score since it mixed unrelated words, namely JinHai, Love You, One. RING-M got a slightly lower score than RING. In RING-M keywords "Serve the Ball" is about the progress of the game while "Revenge Humiliation" and "Superior To" is about the result of the game. We can see RING-M, *i.e.,* modularity based method, mixed keywords of two sub-events.

#### 5.2.4 Keywords Summarization Quality

Here we evaluate the precision and recall of the keyword summarization of events. For the 17 events detected by all methods, we manually label them with a representative tweet from authoritative publishers as human labeled refer-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



Table 5: Case Study: Tennis Game Progress and Result, Roger Federer vs. Djokovic

| Method | Keywords | | | NPMI | | Rouge-P | | Rouge-R | | Rouge-F | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RING | Roger Federer | Service Break | Win, Tennis, Not Sure, Serve the Ball, Save, Tight Game | 0.7836 | 0.8482 | 0.5 | 0.75 | 0.3704 | 0.5 | 0.4255 | 0.6 |
| | | | Djokovic, Win, Tennis, Not Sure, Serve the Ball, Save | | 0.8336 | | 0.625 | | 0.4167 | | 0.5 |
| | | Finals | Straight Win | | 0.7456 | | 1 | | 0.1777 | | 0.2857 |
| | | | ATP Masters 1000, Gilles Simon, Djokovic, Win, Superior To, Tennis, Triple Champion | | 0.8806 | | 0.6667 | | 0.3704 | | 0.4761 |
| RING-M | Serve the Ball, ATP Masters 1000, Revenge Humiliation, Roger Federer, Superior To | | | 0.7409 | | 0.2272 | | 0.1852 | | 0.2401 | |
| KeyGraph | Semi-Finals, Djokovic, ATP Masters 1000 | | | 0.9237 | | 1 | | 0.1111 | | 0.2 | |
| Bitrem | Shanghai, Finals, ATP Masters 1000, JinHai, Love You, China, One, Game,Gilles Simon | | | 0.3873 | | 0.5 | | 0.1852 | | 0.2703 | |

ence summarization. Sub-event summarizations are generated in the same manner.

We choose ROUGE-N as our evaluation metric, which is defined below:

$$\frac{\sum_{S\in\{Ref\ Summaries\}}\sum_{gram_n\in S} Count_{match}(gram_n)}{\sum_{S\in\{Ref\ Summaries\}}\sum_{gram_n\in S} Count(gram_n)} \quad (8)$$

$n$ stands for the length of n-gram. $gram_n$ and $Count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries [52]. In our scenario, we choose $N = 1$ since it is proven to perform great in evaluating very short summaries [49]. The results are shown in Table 6.

**Ring Analysis.** We can see that RING has the best overall performance. This shows that the trending keywords selected by our method are more representative and complete than results of comparing methods. The advantage of RING as analyzed in NPMI evaluation also applies to ROUGE-1 score. For sub-event summarization quality, the score has a higher precision and a slightly lower recall. This indicates that for sub-events (1) the keywords tend to be more precise (2) the keywords show limited aspects of the event. This result is consistent with our design and shows effectiveness of sub-event detection method.

Table 6: ROUGE-1 Results

| Method | | ROUGE-1 | | |
|---|---|---|---|---|
| | | Precision | Recall | F-Score |
| RING-M | | 0.550 | 0.373 | 0.444 |
| RING | Event($k$=3) | 0.553 | **0.396** | **0.462** |
| | Event($k$=4) | 0.626 | 0.390 | **0.481** |
| | Sub-Event | 0.627 | 0.360 | 0.457 |
| KeyGraph | | 0.317 | 0.124 | 0.179 |
| Biterm | | **0.635** | 0.210 | 0.315 |

**Comparison Analysis.** Compared to RING-M, keyword summarization of RING has a better precision and recall score. This is because RING tends to include more precise keywords and many aspects of the event. Though RING-M has a similar performance to RING, since it can not differentiate sub-events of events, it leads to difficulty in human comprehension as shown in Table 5.

KeyGraph tends to miss the trending keywords detected by RING and include unrelated keywords, leading to poor performance. Biterm has a better precision and a lower recall than RING. This, at first sight, seems that RING finds many related keywords but brought in more unrelated words. This is because keywords from other aspects of event results of RING are considered as unrelated to the reference summarization, leading to drop in precision in ROUGE-1 scores. From the NPMI results based on the original document collection data, we can see that RING produces a more coherent result, proving that the keywords found by RING are representative and correlated to the detected event. While in ROUGE-1 experiment, we use a single tweet as reference summarization. This might fall short as partial since events might include many aspects, which will be proved in the following case study and Section 4.3.6. Biterm would not include as many aspects of the event, leading to a lower recall score.

**Parameter Sensitivity.** Here we evaluate the effects of parameter $k$. We can see that precision is improved due to stronger coherence. It may also come from reduced aspects of events against the partial reference summarization. Recall is reduced since some aspects of the event might be filtered out due to a more strict community definition.

**Case Study.** Here we study the ROUGE scores in Table 5. RING's precision score on the event proves that a single tweet might fall short as a partial summary for a whole event. RING's high recall score proves that the method can effectively select representative keywords. RING-M mixed events together and did not find sufficient keywords for the events, hence the poor performance. KeyGraph only selected 3 keywords for the whole event, which are all correct but clearly not enough. Biterm mixed unrelated keywords and did not find sufficient keywords for the event.

Sub-event results evaluate summary performance specific to one aspect of the event. We see results for the first two "same" sub-events, they actually could represent different opinions. In the second sub-event people may be talking more about if Djokovic would save the game while the first sub-event may be more about the "tight game". We used the same reference summarization for them. For the third sub-event with only 3 keywords, the precision is high but it does not get sufficient keywords. Words "superior to", "Triple Champion" and "Win" of the fourth event did not appear in the summarization hence the performance.

The duplicated keywords of different sub-events in the case study shows that allowing keywords overlap would yield a more natural result from a sub-event perspective, the lack of which would bring down the performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data



Table 7: A Case of Event Evolution Chain

| Time | Event description |
|---|---|
| 2014-12-01 18:40:00 | The suspect of Fudan poisoning case writes an apology letter to the victim's parents. \| The $2^{nd}$ trial will be held. |
| 2014-12-08 08:10:00 | Fudan poisoning case's second trial will be held in 10am today, victim's parents will be in court. |
| 2014-12-08 12:50:00 | LIVE: Defendant of Fudan poisoning case cries in court. |
| 2015-01-08 07:40:00 | The court will pronounce judgement of the $2^{nd}$ trial today and victim's father hope to maintain the death penalty. |
| 2015-01-08 10:30:00 | The court maintains the former death sentence on attempted murder in the second trail of Fudan poisoning case. |

### 5.3 Event Evolution Monitoring

We demonstrate through case study the effectiveness of RING for multi-granularity, multi-aspect event monitoring that it can tell the story of an event automatically. An evolution chain distilled by production RING system is shown in Table 7. Given (by search or trending event ranking interface) the "latest" event in the table, an evolution chain that spans several month was revealed. These events are about the second trial of the infamous poisoning case in Fudan University, China, where the suspect poisoned his roommate to death in April 2013. The detected events are all significant in its development and we can see that forecast, process and judgement of the trial all drew a lot of attention, hence detected by RING. The descriptions of these events are extracted automatically either directly from tweets or from representative tweets that contained predefined brackets indicating proper titles. Recall from 3.1 that our system reports trending events in 10-minute intervals.

Each of these events in Table 7 has been repeatedly detected in consecutive time intervals, lasting from 20 minutes to 60 minutes. It is indicated by trending keywords detection algorithm that a consecutive exponential growth of tweets in volume has been detected. The system consolidate such similar events to show only one such event and highlight such event for special attention. We manually inspect the average latency of detection of these events, i.e. the time from the first tweet appearing in our data to the detection time, to be 14.4 minutes, which is shown in the table.

### 5.4 Efficiency of Anomaly Detection

We now demonstrate the efficiency and linear horizontal scalability of our approach using data on Sep.26.2014, which contains 9808056 tweets. In RING system, trending keyword detection is run on Spark cluster while event detection and analysis are run on a single node. There are 7 worker nodes and 1 master node in the Spark cluster. We set Spark parameter *driver-memory* and *executor-memory* to $30GiB$ and set *executor-cores* to 8.

**Effects of Data Volume.** We discover that the processing efficiency of trending keyword detection is affected by data volume in each time slice on Spark. To simulate the full data volume of Weibo, we duplicate Sep.26 data 10 and 20 fold to generate larger datasets. We see from Figure 4 that processing efficiency of the system increases with data volume. Particularly, the system can process 16,000 tweets per second with 20-fold of data, which is a 10X speed up compared to speed at 1N. Since phase II consumes only a small portion of the run time, we estimate that the system is competent to process the entire Weibo and Twitter stream, which are at the scale of 20 - 50 million tweets per day. Such acceleration comes from increased ratio of effective computation using MapReduce.

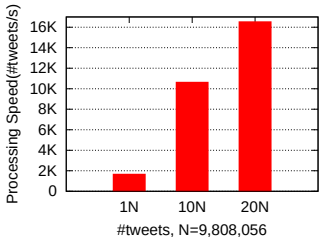
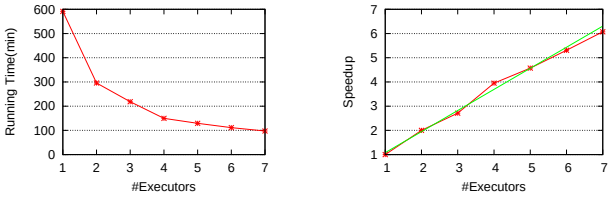



Figure 4: Effects of Data Volume




**(a)** Running Time **(b)** Speed Up

Figure 5: Horizontal Scalability

**Horizontal Scalability.** Figure 5 shows the result when we use different number of worker nodes for phase I (by setting parameter *num-executors* of Spark to 1 through 7). We see that the processing speed increases linearly with the number of computing nodes, demonstrating the horizontal scalability of our graph stream model.

### 5.5 Indexing System Evaluation

To demonstrate the efficiency of the RINGINDEX system and the effectiveness of our strategy, we experimented with the following 4 parameters in the indexing system to show their effects. *TR* is defined as the length of time range. *sub* is defined as the count of sub-partitions in each time range. *SW* is defined as the length of querys time range. *count* is defined as the the count of the querys terms. The strategy that does not update old indices is used as the baseline. We call it *No Merge*, which comes as default for elasticsearch.

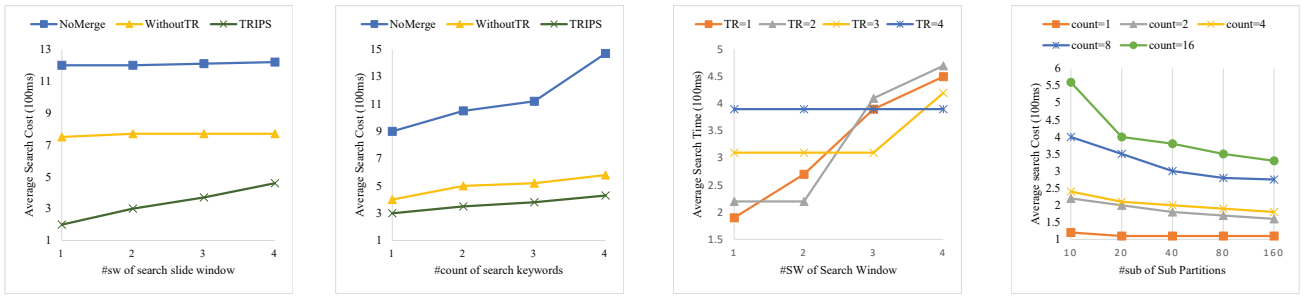
#### 5.5.1 *Efficiency of Searching.*

In this experiment, the keyword queries are generated by selecting terms from a fixed glossary with 201,196 terms at random. Given a query with specific time range, the indexing system uses the time range structure (*TR*) and sub-partition (*sub*) to make a quick navigation. In our experiments, the "without TR" means that the indexing system uses sub-partitions only and "without sub" uses time range structures only.

Figure 6 (a) and 6 (b) shows the time efficiency of searching for different SW and count respectively. Since "No Merge" index and "without TR" (time range) don't have the time range structure, they have stable time cost when the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2672672, IEEE Transactions on Big Data






**(a)** cost on different *DW* **(b)** cost on different *count* **(c)** cost on different *SW* and *TR* **(d)** cost on different *count* and *sub*

Figure 6: Efficiency of RINGINDEX Searching

SW (search window) increasing. As the number of terms increases in the query, search cost would increase for all methods. An important observation is that our complete solution (TRPIS) has the best searching performance. Figure 6 (c) shows the relationship between TR and SW. We can see that if SW is smaller than TR, the cost will be constant as the system would read the same index each time. And for a fixed SW, an equal TR would yield the best performance. Figure 6 (d) illustrates the average search time cost for different sub and count. Basically, the shorter the query, and finer the sub-partitions are, the better the performance. We can see that the time ranged query performance has improved up to 10X in RING compared to elasticsearch.

### 5.6 Graph System Evaluation

RINGGRAPH has been evaluated on three different kinds of tasks, namely improvement of graph update, incremental algorithms and hotspot reblancer. We benchmark RINGGRAPH against the original GraphX. The graph partition number was set as 32 ($4 \times 8$ in the hash-based partitioning method), and the results of all experiments are averages of ten runs. A cluster of five machines are used in the testbed environment, one as master node and other four as workers.

#### 5.6.1 *Graph Update Efficiency.*

To demonstrate graph update efficiency, three different real-world graph datasets are used: (1) **Google Web Graph.** It consists of 875K nodes and 5.1M edges. Nodes represent web pages, and directed edges represented hyperlinks between them. (2) **LiveJournal friendship social network.** It consists about 4.8M nodes and nearly 68.9M edges. Nodes represent users of LiveJournal, and directed edges represent friendship between them. (3) **Friendster friendship social network.** It consists of about 65.6M nodes and nearly 1.8B edges. Nodes represent users of Friendster, and directed edges represent friendship between them.

We generate 10,000 and 100,000 messages per batch to simulate real graph stream update scenarios. The update messages consist of three types of updates, *i.e.*, update of vertex attribute, edge attribute and insertion of new nodes and edges, at a ratio of 25%, 25% and 50%. RINGGRAPH had a speed up of 3.7X, 7.1X and 14.3X respectively compared to GraphX due to finer granularity of updates.

#### 5.6.2 *Incremental Processing.*

Incremental PageRank is used to test the performance improvement of the RING system over the GraphX's default version. Incremental algorithms could reuse the previous results of the last iteration, the computing time and network traffic can be greatly reduced. The larger the graph, the more time will be used to recompute PageRank in the default non-incremental version. With the increased size of the three datasets, the compute time of incremental PageRank on average has speedups of 4.3X, 6.3X and 16.7X on 32 cores. Network traffic decreases by 80%, 88%, 90% on 32 cores.

#### 5.6.3 *Effectiveness of Workload Rebalancer.*

We do this experiment running event monitoring algorithms on the Weibo dataset. With the help of the load balancing technique, the cost of time to run event monitoring algorithm on the whole dataset is reduced by 11%.

## 6 RELATED WORK

In this section, we review related work about topic detection and emerging event detection using keyword co-occurrence.

It has long been recognized that modeling topics or events based on keyword co-occurrence is an effective approach. Co-occurrence information has been used for term clustering [53] and keyword extraction from documents [54]. Palla et al. [28] discovered that a word's contexts of different meanings could be represented as overlapping communities in the word co-occurrence graph. A short text topic model that directly models the generation of word co-occurrence pattern has been proposed [15]. Sayyadi et al. addressed event detection from news articles [55] and performed topic detection for large and noisy social media collections [27]. They adopt a betweenness centrality based clustering algorithm over keyword co-occurrence graph for the task. They do not address the problem of real-time emerging event detection from short text stream.

Emerging event detection methods follow similar processes involving trending keyword detection and clustering for event identification. TwitterMonitor [16] uses queuing theory to detect bursty keywords while trends and event analysis are both computed with history data based on co-occurrence relations in an out-of-core fashion. Long et al. [56] utilizes word frequency and entropy to detect bursty keywords and used a hierarchical divisive clustering method to get event clusters. Weng et al. [26] adopts wavelet analysis for bursty keywords detection and modularity-based graph partitioning technique over keyword co-occurrence graph to get event clusters. Schubert et al. [5] uses a scalable outliers detection technique for

bursty keywords detection and a hierarchical divisive clustering method for trend refinement. For Twitter, this method would generate many trends that only contain a single keyword, which is hard to comprehend. The above six methods lack control over clustering resolution and would mix unrelated keywords with correlated ones. Keywords are now allowed to overlap between topics and sub-events and event correlation analysis are both absent.

In contrast, our approach adopts a unified graph processing framework through each phase and meets all the listed semantic requirements. Interestingly, later work of TwitterMonitor [57], [58] assumes the existence of such a framework, and firstly studies dense subgraph maintenance problem under streaming edge weight update, where the graph is composed of entities and their co-occurrence relations. Detecting emerging events basically requires recomputing clustering from scratch while the efficiency of [57] largely exploits incremental computation. As shown in their experiment, the proposed algorithm requires each detected subgraph to have no more than 5 nodes and increasing this number would lead to the significant drop of efficiency [57]. In the scenario, the number of keyword nodes is much larger than 5. Hence method [57] does not fit in terms of efficiency. What's more, entity recognition is costly for streaming setting in Chinese.

All these existing work neither adopts the distributed stream computing paradigm, nor provides the event correlation analysis into their consideration.

## 7 Conclusions

We have demonstrated RING, a real-time emerging event monitoring system over microblog platforms, that integrates our efforts on both emerging event monitoring research and system research. RING is able to monitor emerging event as to detect emerging events, build event correlations and trace event evolutions. Further, RING's infrastructure is equipped with customized optimization on its full-text search engine and distributed graph processing engine to perform event monitoring more efficiently. What's more RING supports event and text queries and many other functionalities to assist the analysis of emerging events, as demonstrated in the user interface. The system clearly presents its advantages over existing systems and methods from both technical and system perspectives for the emerging event monitoring task. In the future, we would conduct research on automatic story telling and envolving knowledge base into our results to provide better intelligence.

## References


[1] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes twitter users: real-time event detection by social sensors," in *WWW*, 2010.

[2] R. McCreadie, C. Macdonald, I. Ounis, M. Osborne, and S. Petrovic, "Scalable distributed event detection for twitter," in *IEEE BigData*, 2013.

[3] Y. Chen, H. Amiri, Z. Li, and T.-S. Chua, "Emerging topic detection for organizations from microblogs," in *SIGIR*, 2013.

[4] W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K. Wang, "Topicsketch: Real-time bursty topic detection from twitter," in *ICDM*, 2013.

[5] E. Schubert, M. Weiler, and H.-P. Kriegel, "Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds," in *KDD*, 2014.

[6] F. Wei, S. Liu, Y. Song, S. Pan, M. X. Zhou, W. Qian, L. Shi, L. Tan, and Q. Zhang, "Tiara: a visual exploratory text analytic system," in *KDD*, 2010.

[7] P. Lee, L. V. Lakshmanan, and E. E. Milios, "Incremental cluster evolution tracking from highly dynamic network data," in *IEEE International Conference on Data Engineering (ICDE)*, 2014, pp. 3–14.

[8] C. Li, A. Sun, and A. Datta, "Twevent: segment-based event detection from tweets," in *CIKM*, 2012.

[9] D. Metzler, C. Cai, and E. Hovy, "Structured event retrieval over microblog archives," in *HLT-NAACL*, 2012.

[10] C. Budak, T. Georgiou, and D. A. A. El Abbadi, "Geoscope: Online detection of geo-correlated information trends in social networks," *PVLDB*, vol. 7, no. 4, 2013.

[11] J. Allan, R. Papka, and V. Lavrenko, "On-line new event detection and tracking," in *SIGIR*, 1998.

[12] T. Hofmann, "Probabilistic latent semantic analysis," in *UAI*, 1999.

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[14] D. M. Blei and J. D. Lafferty, "Dynamic topic models," in *ICML*, 2006.

[15] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," in *WWW*, 2013.

[16] M. Mathioudakis and N. Koudas, "Twittermonitor: trend detection over the twitter stream," in *SIGMOG*, 2010.

[17] R. Xie, F. Zhu, H. Ma, W. Xie, and C. Lin, "Clear: A real-time online observatory for bursty and viral events," *PVLDB Demo*, 2014.

[18] H. Cai, Z. Huang, D. Srivastava, and Q. Zhang, "Indexing evolving events from tweet streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 11, pp. 3001–3015, 2015.

[19] C. Chen, F. Li, B. C. Ooi, and S. Wu, "Ti: an efficient indexing mechanism for real-time search on tweets," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 649–660.

[20] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin, "Earlybird: Real-time search at twitter," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 1360–1369.

[21] Y. Shao, B. Cui, and L. Ma, "Page: a partition aware engine for parallel graph computation," *IEEE Transactions on Knowledge and Data Engineering*, 2015.

[22] 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 *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[23] "Ring," http://ring.cnbigdata.org/. [Online]. Available: http://ring.cnbigdata.org/

[24] W. Yu, C. C. Aggarwal, S. Ma, and H. Wang, "On anomalous hotspot discovery in graph streams," in *ICDM*, 2013.

[25] L. Gao, C. Song, Z. Gao, A.-L. Barabási, J. P. Bagrow, and D. Wang, "Quantifying information flow during emergencies," *Nature Scientific Reports*, vol. 4, 2014.

[26] J. Weng and B.-S. Lee, "Event detection in twitter." *ICWSM*, 2011.

[27] H. Sayyadi and L. Raschid, "A graph analytical approach for topic detection," *ACM TOIT*, 2013.

[28] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, 2005.

[29] F. Reid, A. McDaid, and N. Hurley, "Percolation computation in complex networks," in *ASONAM*, 2012.

[30] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *DMKD*, vol. 8, no. 1, pp. 53–87, 2004.

[31] Z. Lu, W. Yu, R. Zhang, J. Li, and H. Wei, "Discovering event evolution chain in microblog," in *IEEE HPCC*, 2015.

[32] P. Lee, L. V. Lakshmanan, and E. Milios, "Cast: A context-aware story-teller for streaming social content," in *CIKM*, 2014.

[33] A. Saha and V. Sindhwani, "Learning evolving and emerging topics in social media: a dynamic nmf approach with temporal regularization," in *WSDM*, 2012.

[34] H. Liu, Y. Zhang, H. Lin, J. Wu, Z. Wu, and X. Zhang, "How many zombies around you?" in *ICDM*, 2013.

[35] H. Huang, J. Li, R. Zhang, W. Yu, and W. Ju, "Liveindex: A distributed online index system for temporal microblog data," in *IEEE HPCC*, 2015.

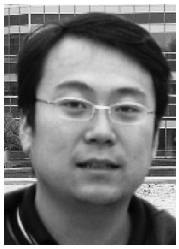
[36] Y. Shao, J. Yao, B. Cui, and L. Ma, "Page: A partition aware graph computation engine," in *Proceedings of the 22nd ACM international*

*conference on Conference on information & knowledge management*. ACM, 2013, pp. 823–828.
[37] 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 *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.
[38] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*. ACM, 2013, p. 22.
[39] R. Power and J. Li, "Piccolo: Building fast, distributed programs with partitioned tables." in *OSDI*, vol. 10, 2010, pp. 1–14.
[40] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.
[41] R. Pearce, M. Gokhale, and N. M. Amato, "Multithreaded asynchronous graph traversal for in-memory and semi-external memory," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.
[42] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 2009, pp. 229–238.
[43] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 17–30.
[44] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: graph processing at facebook-scale," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.
[45] D. Yan, J. Cheng, Y. Lu, and W. Ng, "Blogel: A block-centric framework for distributed computation on real-world graphs," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 1981–1992, 2014.
[46] Y. Zhang, X. Liao, H. Jin, L. Lin, and F. Lu, "An adaptive switching scheme for iterative computing in the cloud," *Frontiers of Computer Science*, vol. 8, no. 6, pp. 872–884, 2014.
[47] Ü. i. t. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 656–683, 2010.
[48] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, no. 10, 2008.
[49] D. Newman, J. H. Lau, K. Grieser, and T. Baldwin, "Automatic evaluation of topic coherence," in *HLT-NAACL*, 2010.
[50] C. Wang, M. Danilevsky, J. Liu, N. Desai, H. Ji, and J. Han, "Constructing topical hierarchies in heterogeneous information networks," in *IEEE International Conference on Data Mining (ICDM)*, 2013, pp. 767–776.
[51] G. Bouma, "Normalized (pointwise) mutual information in collocation extraction," in *Biennial GSCL Conference*, 2009.
[52] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *ACL Workshop: Text Summarization Branches Out*, 2004.
[53] F. Pereira, N. Tishby, and L. Lee, "Distributional clustering of english words," in *ACL*, 1993.
[54] Y. Matsuo and M. Ishizuka, "Keyword extraction from a single document using word co-occurrence statistical information," *IJAIT*, vol. 13, no. 01, pp. 157–169, 2004.
[55] H. Sayyadi, M. Hurst, and A. Maykov, "Event detection and tracking in social streams." in *ICWSM*, 2009.
[56] R. Long, H. Wang, Y. Chen, O. Jin, and Y. Yu, "Towards effective event detection, tracking and summarization on microblog data," in *WAIM*, 2011.
[57] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *PVLDB*, pp. 574–585, 2012.
[58] A. Angel, N. Koudas, N. Sarkas, D. Srivastava, M. Svendsen, and S. Tirthapura, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *VLDBJ*, vol. 23, no. 2, pp. 175–199, 2014.

**Weiren Yu** received the BE degree from the School of Advanced Engineering at Beihang University, China in 2011. He is currently a PhD candidate in the Department of Computer Science, Beihang University since 2011. His research interests include distributed machine learning systems, scalable graphical models and graph mining models for emerging event detection on social media.

**Jianxin Li** is an associate professor at the School of Computer Science and Engineering, Beihang University, China. He received his PhD degree from Beihang University in 2008. He was a visiting scholar in machine learning department of Carnegie Mellon University, USA in 2015, and a visiting researcher of MSRA in 2011. His current research interests include data analysis and processing, distributed systems, and system virtualization.

**Md Zakirul Alam Bhuiyan** received the PhD degree and the M. Eng degree from Central South University, China, in 2009 and 2013 respectively, and the BSc degree from International Islamic University Chittagong, Bangladesh, in 2005, all in Computer Science and Technology. He is currently an assistant professor of the Department of Computer and Information Sciences at the Fordham University. Earlier, he worked as an assistant professor (research) at the Temple University and a post-doctoral fellow at the Central South University, China, a research assistant at the Hong Kong PolyU, and a software engineer in industries. His research focuses on dependable cyber physical systems, WSN applications, network security, and sensor-cloud computing. He has served as a managing guest editor, general chair, program chair, workshop chair, publicity chair, TPC member, and reviewer of international journals/conferences. He is a member of IEEE and a member of ACM.



**Richong Zhang** received his BS degree and MASc degree from Jilin University, China in 2001 and 2004, respectively. In 2006, he received his MS degree from Dalhousie University, Canada. He received his PhD form the School of Information Technology and Engineering, University of Ottawa, Canada in 2011. He is currently an associate professor in the School of Computer Science and Engineering, Beihang University, China. His research interests include artificial intelligence and data mining.

**Jinpeng Huai** is a professor at the School of Computer Science and Engineering, Beihang University, China. He received his Ph.D. degree in computer science from Beihang University, China, in 1993. Prof. Huai is an academician of Chinese Academy of Sciences and the vice honorary chairman of China Computer Federation (CCF). His research interests include big data computing, distributed systems, virtual computing, service-oriented computing, trustworthiness and security.