CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3657

SPECIAL ISSUE PAPER

Bursty event detection from microblog: a distributed and incremental approach

Jianxin Li*,†, Jianfeng Wen, Zhenying Tai, Richong Zhang and Weiren Yu

State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing, China

SUMMARY

As a new form of social media, microblogs (e.g., Twitter and Weibo) are playing an important role in people's daily life. With the rise in popularity and size of microblogs, there is a need for distributed approaches that can detect bursty event with low latency from the short-text data stream. In this paper, we propose a distributed and incremental temporal topic model for microblogs called Bursty Event dEtection (BEE+). BEE+ is able to detect bursty events from short-text dataset and model the temporal information. And BEE+ processes the post-stream incrementally to track the topic drifting of events over time. Therefore, the latent semantic indices are preserved from one time period to the next. In order to achieve real-time processing, we design a distributed execution framework based on Spark engine. To verify its ability to detect bursty event, we conduct experiments on a Weibo dataset of 6,360,125 posts. The results show that BEE+ can outperform the baselines for detecting the meaningful bursty events and track the topic drifting.

Received 4 February 2015; Revised 4 June 2015; Accepted 8 August 2015

KEY WORDS: social network; event detection; temporal topic model; topic drifting

1. INTRODUCTION

Bursty Event dEtection (BEE+) from microblogs is now a hot area in the field of data mining and knowledge discovery, because the information posted in microblogs is real time and often event driven [1]. In microblogs, with the textual content coupled with the temporal patterns, one could discover public's general interest by detecting those bursty events. There is thus an urgent need to provide users with tools that can automatically extract significant information from highly dynamic social streams [2]. However, unlike the traditional normal documents (e.g., news articles and academic papers), the lack of rich context in short texts [3] and the topic of event drifting over time make online event detection from microblogs a challenging problem.

A part of existing studies on detecting bursty events are based on detecting bursty keywords. In general, the bursty events are represented by a group of related bursty keywords (or segmentation) with the bursty temporal information. Some clustering methods, such as graph partitioning and density-based K-means, are used to detect events. The relation of each two keywords can be calculated using the co-occurrences of the words, and the distance predefined of vectors is used to measure the similarity of each two posts. For example, Mathioudakis and Koudas [4] detect events by grouping the bursty keywords. However, the aforementioned methods are not built upon robust probabilistic background. And they have ignored two classical problems called synonymy and polysemy [5]. Moreover, they cannot model the topic drifting of events.

*Correspondence to: Jianxin Li, School of Computer Science and Engineering, Beihang University, Beijing, China.
†E-mail: lijx@act.buaa.edu.cn

Other popular methods to detect emergency events are based on topic models, like probabilistic latent semantic analysis (PLSA) [6] and Latent Dirichlet Allocation (LDA) [7], which are widely used to uncover the hidden variables from the text collections. In this case, the occurrence of words can be modeled with the probabilistic theory, and the problems of synonymy and polysemy have also been solved. But those conventional topic models reveal the latent variables within the text corpus by implicitly capturing the document-level word co-occurrence patterns [8, 9], which made them not applicable to microblogs posts. Because direct application of these models on microblogs data stream will suffer from the severe data sparsity problem we mentioned earlier (i.e., the sparse word co-occurrence patterns in each short document) [10], the drifting of event topic over time cannot be measured and the temporal information cannot be modeled.

In this paper, based on the topic model for short text, we proposed before [11], we designed a new distributed and incremental temporal topic model (BEE+) for detecting bursty events from microblogs. BEE+ can model the temporal information of events in microblogs and identify the bursty events based on the temporal information. And BEE+ employs processing the post-stream incrementally, so it can track the topic of events drifting over time. So the latent semantic indices are preserved from one time period to the next. At the same time, with the incremental topic model BEE+, we applied a delicate parameter estimating process, which has a faster convergence time than the traditional topic model. To verify BEE+'s ability to detect event, we applied our BEE+ models on a dataset of 6,360,125 posts. The experimental results show that BEE+ can detect more meaningful events than the baselines and depict topic drifting well. And the execution time with different sizes of dataset and numbers of executors shows our model's efficiency.

The remainder of this paper is organized as follows. In Section 2, we describe the BEE+ model. Section 3 details the parameters inference. The distributed implementation is illustrated in Section 4. The experimental results are presented in Section 5. In Section 6, we review briefly the related work. Finally, we conclude the paper in Section 7.

2. BURSTY EVENT DETECTION: TOPIC MODEL FOR SHORT TEXT

2.1. Notation and problem formulation

We first introduce the notations used in this paper and formally formulate our problem. All the variables are explained in Table I. We assume that we have a stream of D microblogs posts, denoted as $d_1, d_2, \ldots, d_D$. Each post d_i is also associated with a discrete timestamp $t_d = t$, where t is an index between 1 and T. T is the total number of time points we consider. Each document d_i is denoted as a bag of words $\{w_{i,1}, w_{i,2} \ldots w_{i,j} \ldots w_{i,N_i}\}$, where j is an index between 1 and V and V is the size of the vocabulary. N_i is the number of the words in document d_i.

A bursty event is a topic that (1) is a surge in the number of posts related to this topic and (2) popular in the posts during the period when it surges. Given the post-stream D, our task in this paper is to detect the bursty events from D by using an online algorithm and implementation.

Table I. Notations of BEE+ model.

Variable	Signification	Variable	Signification
d	One document	$\vec{w}$	One vector of words
z	One topic	H	All documents with hashtag
t	One time stamp	$\tilde{H}$	All documents without hashtag
h	One hashtag	λ_θ	Probability of background topic
w	One word	θ	Background topic
$n(w,d)$	Frequency of w in d		

BEE+, Bursty Event dEtection.

DOI: 10.1002/cpe

2.2. The generative process of Bursty Event dEtection

The basic ideas of our BEE+ model are as follows. 1) The posts published around the same time is more likely to be about the same topic than the other random posts. 2) One post is always related to one event, we will introduce a background topic later in this section. 3) The posts with the same hashtag are always related to the same topic. Note that not all posts contain a hashtag, so BEE+ presents the hashtag with a missing value for the posts without hashtag labels.

In BEE+, we assume that there are K latent topics in the text stream, d denotes a document, w denotes a term in a document, z denotes a latent variable, and t_d denotes the released time of document d. In order to explain all the words in the stream, including the meaningless words and the common words, we further propose a background topic, with a distribution θ over common words. All the words in each post are generated from some mixture of these $K+1$ underlying topics.

In standard topic models, such as PLSA and LDA, a document is generated by a mixture of topics, represented by a distribution over topics. This is a reasonable assumption for long documents. But in microblogs, the posts often contain only one or several sentences. Consequently, a single post is most likely to be about one event or meaningless. We therefore associate a single hidden variable and the background topic with each post to indicate its topic. The assumption of assigning a single topic to a short sequence of words has been used before [1, 12, 13]. As we have discussed in Section 1, the posts published around the same time are more likely to be about the same topic than the other random posts with the similar idea in [9]. To model this observation, we assume that there is a distribution over time points for each topic. The higher the probability for a topic at one time point is, the more popular the topic is at that time.

The generative process for each document d in D is as follows.

1. Choose N from the Poisson distribution.
2. Choose t_d from multinomial distribution.
3. If this document has a hashtag,
   a. choose a topic from $p(z|d, t_d, h)$;
   b. from 1 to N, choose a word from $p(w|z)$ to $p(w|\theta)$.
4. If this document has no hashtag,
   a. choose a topic from $p(z|d, t_d)$;
   b. from 1 to N, choose a word from $p(w|z)$ to $p(w|\theta)$).

3. PARAMETERS INFERENCE

3.1. Initial inference

Figure 1 shows the graphical model representation of BEE+, the joint distribution of the word vector $\vec{w}$ of each document d is

$$p(\vec{w}, d) = \begin{cases} p(d)p(t_d)p(h_d)\sum_z p(z|d,t_d,h_d)\prod_{w\in d}[\lambda_\theta p(w|\theta) + (1-\lambda_\theta)p(w|z)]^{n(w,d)} & d \in H \\ p(d)p(t_d)\sum_z p(z|d,t_d)\prod_{w\in d}[\lambda_\theta p(w|\theta) + (1-\lambda_\theta)p(w|z)]^{n(w,d)} & d \in \tilde{H} \end{cases}$$

where λ_θ is the mixture weight of the background topic, which appears in the stream dataset with the same intensity all the time. Note that d, h, and t are coupling.

So, the probability of the complete corpus in one window is as follows.

$$p(\vec{w}, \vec{D}) = \begin{cases} \prod_d \prod_t \prod_h p(d)p(t)p(h)\sum_z p(z|d,t,h)\prod_{w\in d}[\lambda_\theta p(w|\theta)+(1-\lambda_\theta)p(w|z)]^{n(w,d)} & d \in H \\ \prod_d \prod_t p(d)p(t)\sum_z p(z|d,t)\prod_{w\in d}[\lambda_\theta p(w|\theta) + (1-\lambda_\theta)p(w|z)]^{n(w,d)} & d \in \tilde{H} \end{cases}$$

The log-likelihood of the generating text sample is thus

DOI: 10.1002/cpe

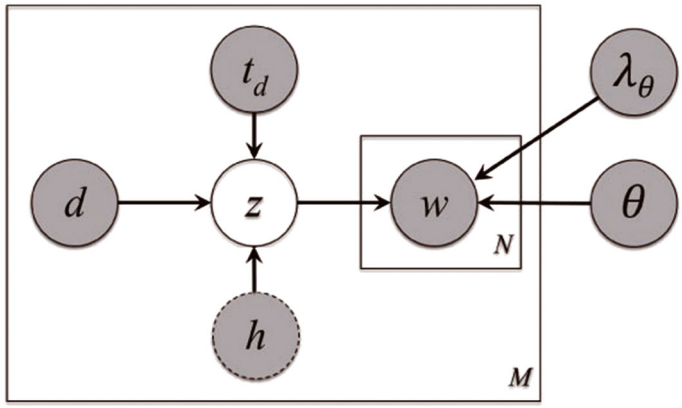

Figure 1. Graphical model representation of Bursty Event dEtection.

$$log\mathcal{L} = \sum_{t}\left\{\sum_{d\in\tilde{H}} log\left\{p(d)p(t)\sum_{z}p(z|d,t)\prod_{w}[\lambda_\theta p(w|\theta)+(1-\lambda_\theta)p(w|z)]^{n(w,d)}\right\}\right.$$
$$\left.+\sum_{d\in H}\sum_{h} log\left\{p(d)p(t)p(h)\sum_{z}p(z|d,t,h)\prod_{w}[\lambda_\theta p(w|\theta)+(1-\lambda_\theta)p(w|z)]^{n(w,d)}\right\}\right\} \tag{1}$$

Parameter estimation. The standard procedure for maximum likelihood estimation in latent variable models is the expectation maximization (EM) algorithm [6]. To maximize log-likelihood function, we introduce the estimation of latent topic variable into our log-likelihood function firstly

$$\begin{aligned} log\mathcal{L} = \sum_{t}\Bigg\{\sum_{d\in\tilde{H}} log\Bigg\{p(d)p(t)\sum_{z}\frac{p(z|\vec{w},d,t)}{p(z|\vec{w},d,t)}p(z|d,t) \\ \prod_{w}[\lambda_\theta p(w|\theta)+(1-\lambda_\theta)p(w|z)]^{n(w,d)}\Bigg\} \\ +\sum_{d\in H}\sum_{h} log\Bigg\{p(d)p(t)p(h)\sum_{z}\frac{p(z|\vec{w},d,t,h)}{p(z|\vec{w},d,t,h)}p(z|d,t,h) \\ \prod_{w}[\lambda_\theta p(w|\theta)+(1-\lambda_\theta)p(w|z)]^{n(w,d)}\Bigg\}\Bigg\} \end{aligned} \tag{2}$$

The estimation of latent topic variable is calculated by the expectation (E) step [6]. In our model, according to the Bayes's rule, the conditional probability of $p(z|\vec{w},d,t_d)$ can be estimated by the following in the **E (expectation)** step

$$\begin{aligned} p(z|\vec{w},d,t) &= \frac{(1-\lambda_\theta)p(\vec{w}|z)p(z|d,t)}{\lambda_\theta p(\vec{w}|\theta)+(1-\lambda_\theta)\sum_z p(\vec{w}|z)p(z|d,t)}, d\in\tilde{H} \\ p(z|\vec{w},d,t,h) &= \frac{(1-\lambda_\theta)p(\vec{w}|z)p(z|d,t,h)}{\lambda_\theta p(\vec{w}|\theta)+(1-\lambda_\theta)\sum_z p(\vec{w}|z)p(z|d,t,h)}, d\in H \end{aligned} \tag{3}$$

where $\vec{w}$ denotes the word vector in the document d and $p(\vec{w}|z)=\prod_w p(w|z)^{n(d,w)}$, where $n(w,d)$ denotes the frequency of the word w in the document d. $p(z|d,t)$ and $p(z|d,t,h)$ have been estimated in the previous maximization (M)-step. And λ_θ and $p(w|\theta)$ have been initialized

DOI: 10.1002/cpe

before the iteration starts. We have merged the regularization operation of $p(w|z)$ with the background topic θ into the E-step.

Then, we propose a lower bound for our log-likelihood function, which is easy to maximum

$$\begin{aligned} log\mathcal{L} \geqslant \sum_t \Bigg\{ \sum_{d\in\tilde{H}} \sum_z p(z|\vec{w},d,t) log \Bigg\{ \frac{p(d)p(t)}{p(z|\vec{w},d,t)} p(z|d,t) \\ \prod_w [\lambda_\theta p(w|\theta) + (1-\lambda_\theta) p(w|z)]^{n(w,d)} \Bigg\} \\ + \sum_{d\in H} \sum_h \sum_z p(z|\vec{w},d,t,h) log \Bigg\{ \frac{p(d)p(t)p(h)}{p(z|\vec{w},d,t,h)} p(z|d,t,h) \\ \prod_w [\lambda_\theta p(w|\theta) + (1-\lambda_\theta) p(w|z)]^{n(w,d)} \Bigg\} \Bigg\} \end{aligned} \tag{4}$$

As we have estimated $p(z|\vec{w},d,t)$ and $p(z|\vec{w},d,t,h)$, we can ignore $\frac{1}{p(z|\vec{w},d,t)}$ and $\frac{1}{p(z|\vec{w},d,t,h)}$ in Equation (4). So the problem turns to maximum

$$\begin{aligned} \sum_t \Bigg\{ \sum_{d\in\tilde{H}} \sum_z p(z|\vec{w},d,t) log \Bigg\{ p(d)p(t)p(z|d,t) \prod_w [\lambda_\theta p(w|\theta) + (1-\lambda_\theta) p(w|z)]^{n(w,d)} \Bigg\} \\ + \sum_{d\in H} \sum_h \sum_z p(z|\vec{w},d,t,h) log \Bigg\{ p(d)p(t)p(h)p(z|d,t,h) \\ \prod_w [\lambda_\theta p(w|\theta) + (1-\lambda_\theta) p(w|z)]^{n(w,d)} \Bigg\} \Bigg\} \end{aligned} \tag{5}$$

with the constraints

$$\begin{aligned} \sum_w p(w|z) = 1 \\ \sum_z p(z|d,t) = 1 \\ \sum_z p(z|d,t,h) = 1 \end{aligned}$$

The Lagrange multipliers method is feasible to solve this problem.

So, in the **M (maximization)** step (M-step), the probabilities $p(w|z)$, $p(z|d,t)$, and $p(z|d,t,h)$ can be estimated, respectively, by the following:

$$\begin{aligned} p(w|z) = \frac{\sum_t \sum_{d\in\tilde{H}} p(z|\vec{w},d,t) n(w,d) + \sum_t \sum_{d\in H} \sum_h p(z|\vec{w},d,t,h) n(w,d)}{(1-\lambda_\theta) \sum_t \{ \sum_w \sum_{d\in\tilde{H}} p(z|\vec{w},d,t) n(w,d) + \sum_w \sum_{d\in H} \sum_h p(z|\vec{w},d,t,h) n(w,d) \}} \\ - \frac{\lambda_\theta}{1-\lambda_\theta} p(w|\theta) \end{aligned} \tag{6}$$

$$\begin{aligned} p(z|d,t) = p(z|\vec{w},d,t) \\ p(z|d,t,h) = p(z|\vec{w},d,t,h) \end{aligned} \tag{7}$$

where the $p(z|\vec{w},d,t,h)$ and $p(z|\vec{w},d,t)$ have been estimated in the previous E-step. $n(w,d)$ denotes the frequency of the word w in the document d.

DOI: 10.1002/cpe

3.2. Incremental update

In the sense of online BEE+, one should take the topic drifting into consideration. That is to say, using the estimated $p(w|z)$ parameters to estimate new parameters $p(z|\vec{w}, q, t, h)$, $p(z|\vec{w}, q, t)$ for a new document q in the data stream is not an effective way in our problem. BEE determines whether a new document is related to an existing event or a new event. Because of the topic of event drifts over time, the old inactive event less likely attracts new documents than a recently active event. To follow this changes, we propose a lookup window (Figure 2) that is popular to limit the time frame that an incoming document can relate to. But the EM algorithm needs to initialize them randomly, which makes the lost of topic drift. When we update the lookup window, we should avoid this. So we designed an incremental algorithm for parameter revising.

3.3. Parameters update

As the time window slides forward, our algorithm removes the out-of-date posts d_{old} and removes the terms w_{old} that are not used in recent posts. So, the parameters $p(w_{old}|z)$, $p(z|\vec{w}, d_{old}, t, h)$, and $p(z|\vec{w}, d_{old}, t)$ are also removed. Subsequently, the remaining parameters must be renormalized proportionally as follows, where w_{rest} are those words rest.

$$p(w|z) = \frac{p(w|z)}{\sum_{w_{rest}} p(w|z)} \tag{8}$$

In this step, new documents and terms are arriving from the data stream. There are two different situations.

1. When there is no new word in those new arriving documents,
   $p(w|z)$ are fixed to be used for estimating $p(z|\vec{w}, d_{new}, t_d)$ and $p(z|\vec{w}, d_{new}, t_d, h)$ in (10).

$$\begin{aligned} p(z|\vec{w}, d, t) &= \frac{(1-\lambda_\theta)p(\vec{w}|z)p(z|d,t)}{\lambda_\theta p(\vec{w}|\theta) + (1-\lambda_\theta)\sum_z p(\vec{w}|z)p(z|d,t)}, d \in \tilde{H} \\ p(z|\vec{w}, d, t, h) &= \frac{(1-\lambda_\theta)p(\vec{w}|z)p(z|d,t,h)}{\lambda_\theta p(\vec{w}|\theta) + (1-\lambda_\theta)\sum_z p(\vec{w}|z)p(z|d,t,h)}, d \in H \end{aligned} \tag{9}$$

2. When there are some new words,

   a. We ignore those new words. In this case, the posterior probability calculated by EM for $p(z|\vec{w}, d_{new}, t_d)$ and $p(z|\vec{w}, d_{new}, t_d, h)$ is

$$\begin{aligned} p(z|\vec{w}, d, t) &= \frac{(1-\lambda_\theta)p(\vec{w}|z)p(z|d,t)}{\lambda_\theta p(\vec{w}|\theta) + (1-\lambda_\theta)\sum_z p(\vec{w}|z)p(z|d,t)}, d \in \tilde{H} \\ p(z|\vec{w}, d, t, h) &= \frac{(1-\lambda_\theta)p(\vec{w}|z)p(z|d,t,h)}{\lambda_\theta p(\vec{w}|\theta) + (1-\lambda_\theta)\sum_z p(\vec{w}|z)p(z|d,t,h)}, d \in H \end{aligned} \tag{10}$$

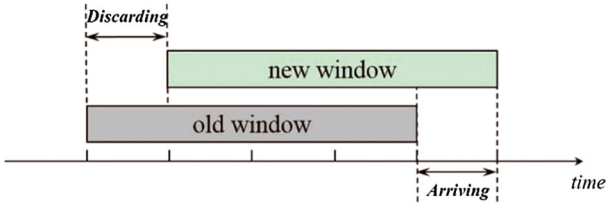

Figure 2. The update process.

DOI: 10.1002/cpe

b. We estimate $p(w_{new}|z)$ for those new words using EM algorithm

$$p(w_{new}|z) = \frac{c(w_{new})}{(1-\lambda_\theta)\sum_{w_{new}} c(w_{new})} - \frac{\lambda_\theta}{1-\lambda_\theta} p(w_{new}|\theta) \tag{11}$$

$$c(w_{new}) = \sum_t \sum_{d \in \vec{H}} p(z|t,d,\vec{w}) n(w_{new},d) + \sum_t \sum_{d \in H} \sum_h p(z|t,d,\vec{w},h) n(w_{new},d);$$

c. Finally, for ensuring the probabilistic rules $\sum_w p(w|z) = 1$, we applied some regularization techniques.

Finally, we reuse the EM algorithm that is represented by Equations (2)–(4), till all the parameters are convergent.

4. DISTRIBUTED IMPLEMENTATION BASED ON SPARK

In this section, we introduce the distributed implementation of our model based on Spark. Our method accomplishes the E-step and the M-step of the *Parameters Revising* section using the map and reduce function of *spark* [14], respectively. But here, we faced two main problems as follows: (1)the result of the E-step is not the final result that we need, but it consumes more storage; and (2) a normalized operation is needed for each step in iteration, which requires another map and reduce function in distributed implementation and increases the computation and communication time. So, we design our data files as four model files, which are $N(w,d)$, $P(w|z)$, $P(z|,d,t,h)$, and $P(z|,d,t)$. The $N(w,d)$ represents the frequency of the word w in the document d. The remaining three files should be manually initialized by other programs before the algorithm starts. These data are the input files for the E-step to calculate $p(z|\vec{w},d,t,h) and P(z|\vec{w},d,t)$. After the calculation in the E-step, these data are recalculated in one iteration of the M-step, and the calculated results are used as the input files of the next iteration. Note that, according to Equation (7), here, we store only $p(z|\vec{w},d,t,h)$ and $p(z|\vec{w},d,t)$ in our implementation.

We use the key-value pair Resilient Distributed Dataset (RDD) to store the $p(w|z)$, $p(z|\vec{w},d,t,h)$, or $p(z|\vec{w},d,t)$ as the input files where the key of the RDD is set to topic index z, and the value is the tuple of the two maps that represent the key-value pairs $p(w|z)$ and $p(z|\vec{w},d,t,h)$ or $p(z|\vec{w},d,t$, respectively. And to speed up the computing, we set the $N(w,d)$ as the broadcast variables. Note that for each topic, one document has a related $p(z|\vec{w},d,t,h)$ or $p(z|\vec{w},d,t$, because all the documents do not have hashtag. Figure 3 is the MapReduce implementation of our algorithm based on Spark.

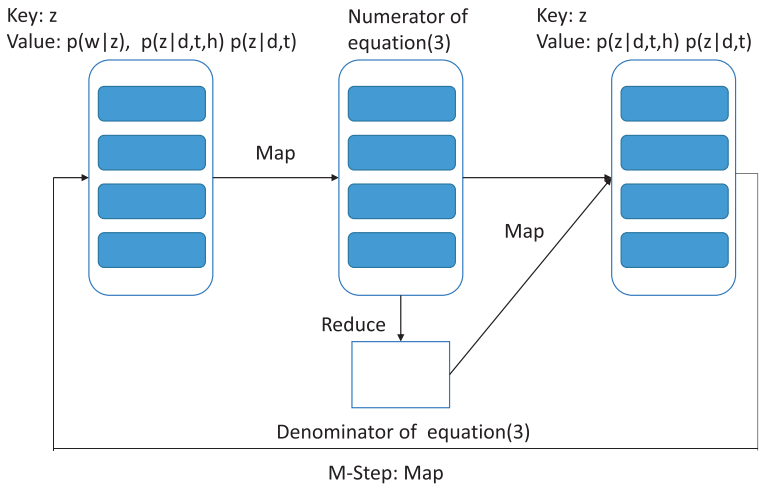

Figure 3. MapReduce implementation of Bursty Event dEtection.

DOI: 10.1002/cpe

4.1. E-step

As described in Algorithm 1. The return files of this function are the posterior probability $p(z|\vec{w},d,t,h)$ or $p(z|\vec{w},d,t)$ key-value pair RDD, where the key represents the document index z and the value is the map from all the $(\vec{w},d,t,h)$ or $(\vec{w},d,t)$ to the corresponding probability of $p(z|\vec{w},d,t,h)$ or $p(z|\vec{w},d,t)$.

Firstly, we calculate the numerator of Equation (3) with a map function of the input RDD data file defined earlier (from lines 1 to 14). Then, because the calculation of the denominator of Equation (3)

Algorithm 1 E-step

Require: key:z; value:$Tuple3 < map < w, val >, map < (\vec{w},d,t,h), val >, map < (\vec{w},d,t), val >>$

Ensure: key:z; value:$Tuple2 < p(z|\vec{w},d,t,h),\ p(z|\vec{w},d,t) >$

1: **RDDNumrator = Map(arg):**
2: **HashMap** $p(\vec{w},z|d,t,h) =$ **New HashMap;**
3: **HashMap** $p(\vec{w},z|d,t) =$ **New HashMap;**
4: **double** $p(\vec{w}|z) = 0$;
5: **For** w in $map < w, val > .keyset$
6: $p(\vec{w}|z) *= p(w|z)^{N(w,d}$)
7: **Endfor**
8: **For** $(\vec{w},d,t,h)$ in $map < (\vec{w},d,t,h), val >$
9: $p(\vec{w},z|d,t,h)$.add($(\vec{w},d,t,h), (1-\lambda_\theta)p(\vec{w}|z) * p(z|\vec{w},d,t,h)$)
10: **Endfor**
11: **For** $(\vec{w},d,t)$ in $map < (\vec{w},d,t), val >$
12: $p(\vec{w},z|d,t)$.add($(\vec{w},d,t), (1-\lambda_\theta)p(\vec{w}|z) * p(z|\vec{w},d,t)$)
13: **Endfor**
14: **return** $key : z; value : Tuple2 < p(\vec{w},z|d,t,h),\ p(\vec{w},z|d,t) >$;

15: **RDDNumrator.Reduce(arg0,arg1):**
16: **HashMap** $p(\vec{w}|d,t,h)$ **= new HashMap**
17: **HashMap** $p(\vec{w}|d,t)$ **= new HashMap;**
18: **For** $(\vec{w},d,t,h)$ in $arg0.\_1.keySet$
19: $p(\vec{w}|d,t,h)$.add($(\vec{w},d,t,h), arg0.\_1.getValue(\vec{w},z|d,t,h) + arg1.\_1.getValue(\vec{w},z|d,t,h)$)
20: **Endfor**
21: **For** $(\vec{w},d,t)$ in $arg0.\_2.keySet$
22: $p(\vec{w}|d,t)$.add($(\vec{w},d,t), arg0.\_2.getValue(\vec{w},z|d,t) + arg1.\_2.getValue(\vec{w},z|d,t)$)
23: **Endfor**
24: **return** $Tuple2 < p(\vec{w}|d,t,h),\ p(\vec{w}|d,t) >$

25: **RDDNumrator.Map(arg):**
26: **HashMap** $p(z|\vec{w},d,t,h)$ **= new HashMap**
27: **HashMap** $p(z|\vec{w},d,t)$ **= new HashMap**
28: **Foreach** $(\vec{w},d,t,h)$
29: **double** $value = \frac{arg.\_1.getValue(\vec{w},d,t,h)}{\lambda_\theta p(\vec{w}|\theta)+p(\vec{w}|d,t,h).getValue(\vec{w}|d,t,h)}$
30: $p(z|\vec{w},d,t,h).add((\vec{w},d,t,h), value)$
31: **Endfor**
32: **Foreach**$(\vec{w},d,t)$
33: **double** $value = \frac{arg.\_1.getValue(\vec{w},d,t)}{\lambda_\theta p(\vec{w}|\theta)+p(\vec{w}|d,t).getValue(\vec{w}|d,t)}$
34: $p(z|\vec{w},d,t).add((\vec{w},d,t), value)$
35: **Endfor**
36: **return** key:z value:$Tuple2 < p(z|\vec{w},d,t,h),\ p(z|\vec{w},d,t) >$;

DOI: 10.1002/cpe

needs a summation for all those topics, we design a reduce function (lines 15–24) for the output RDDNumerator (line 1). Finally, we combined the numerator and denominator together to obtain all the target values $p(z|\vec{w},d,t,h)$ and $p(z|\vec{w},d,t)$ for each z determined by RDD's key. At the end of the E-step, we add an additional regularization to assure that those probabilities result with respect to the probabilistic rules. For now, all the $p(z|\vec{w},d,t,h)$ or $p(z|\vec{w},d,t)$ have been computed. They will be sent to the M-step as the input file.

4.2. M-step

As described in Algorithm 2. The input data are the results from the E-step. The key in the key-value pair is the topic index z, and the value is the pair of map that represents all the probabilities of $p(z|\vec{w},d,t,h)$ and $p(z|\vec{w},d,t)$. In the M-step, firstly, we calculate the numerator as same as in the E-step (lines 4–11). Then, the denominator needs to sum the result in line 11. According to Equation (6), all the elements needed to obtain $p(w|z)$ are contained in RDD's value, so we can implement the E-step in one map function. And Equation (7) permits us to just keep the $p(z|\vec{w},d,t,h)$, $p(z|\vec{w},d,t)$ as the result of $p(z|d,t,h)$, $p(z|d,t)$.

Algorithm 2 M-step

Require: key:z; value:$Tuple2 < p(z|\vec{w},d,t,h),\ p(z|\vec{w},d,t) >$; the pair from E-Step
Ensure: key:z; value:$map < w, val >$

1: **Map:**
2: **HashMap** $P(w|z)$ = **new HashMap**
3: **Foreach** w

4: **double** $p(w|z) = 0$;
5: **Foreach** $(\vec{w},d,t,h)$
6: $p(w|z) += p(z|\vec{w},d,t,h) * N(w,d)$
7: **EndForeach**
8: **Foreach**$(\vec{w},d,t)$
9: $p(w|z) += p(z|\vec{w},d,t) * N(w,d)$
10: **EndForeach**
11: $P(w|z).add(w, p(w|z))$

12: **EndForeach**

13: **double** $denominator = 0$;
14: **Foreach** w
15: $denominator += p(w|z)$;
16: **EndForeach**
17: **Foreach** w in $P(w|z).keySet$
18: $p(w|z) = \frac{p(w|z)}{denominator}$;
19: **EndForeach**
20: **return** key: z value:$Tuple3 < map < w, val >, map < (\vec{w},d,t,h), val >, map < (\vec{w},d,t), val >>$

5. BURSTY DETECTION

Bursty Event dEtection does not generate the bursty events directly, we use the mechanism based on the idea by TwitInfo [15]. Our model generates a set of events E^c, and the probability of each event evolves over time denoted as $p(t_1|e)$, $p(t_1|e) \ldots p(t|e)$, $p(t_i|e)$ denotes the probability of the event e at time t_i. More tweets are related to the event e with a greater $p(t_i|e)$. At each time point i, we will identify such that $p(t_i|e)$ is large relative to the recent $p(t_{i-1}|e)$, $p(t_{i-2}|e)$, ..., not the

DOI: 10.1002/cpe

local maxima among the sequence of all $p(t|e)$. In our framework, the algorithm must determine whether a time point has a relatively large number of posts in it. We take inspiration from congestion control mechanism of TCP, which must determine whether a packet is taking unusually long to be acknowledged and is thus an outlier. To summarize the algorithm, when the algorithm encounters a significant increase in bin count relative to the historical mean, it starts a new window and follows the increase to its maximum. The algorithm ends the peak's window once the bin count returns to the same level it started at or when it encounters another significant increase [15]. We can use online algorithm to implement the processing. After the bursty detection, we rank the events containing the bursty information in the latest time points based on the probability of the event e. For our model, $p(e)$ represents the number of posts related to the topic $p(e)$ there are in the dataset.

6. EXPERIMENTS

We compare the precision and recall of our BEE+ method with the PLSA, TwitterMonitor (TM), and BEE [11] baselines firstly in this section and then illustrate how we can track the topic drifting.

6.1. Dataset

We use a Weibo (a Chinese Twitter) dataset to evaluate our model. This dataset is obtained by a crawler that can obtain five million posts each day on average. We sample the data of December 19, 2014 to evaluate the precision and recall. After filter, there are 6,360,125 posts in total.

6.2. Parameter setting

In this section, we show the parameter setting of our model. Empirically, $\lambda_{\theta B}$ is between 0.05 and 0.1 [16]. In our experiments, we fix our $\lambda_{\theta B} = 0.05$ and set

$$p(w|\theta_B) = \frac{\sum_d n(w,d))}{\sum_w \sum_d n(w,d)} \tag{12}$$

where $p(w|\theta_B)$ denotes the background topic distributed over the vocabulary and $n(w,d)$ represents the frequency of word w in the document d. The parameter $p(w_d|\theta_B)$ in our BEE experiments is constants, because the background topic appears in the stream dataset with the same intensity all the time.

6.3. Ground truth generation

To compare the precision of our model with other alternative models, we perform the effectiveness evaluation in our experiments. For our model and PLSA, we obtain the result of time series data for a number of topics. We apply the same method explained in the bursty detection module to detect the bursty events [17]. We rank the obtained bursty events by the number of tweets (or words in the case of PLSA model or the keywords in each cluster for TM) assigned to the topics and take the top-K events from each model. Because no ground truth is available about all the 'relevant' events, we manually check the events detected by the three models with the real-world events in the news. As the same idea in [18], we labeled the event to be a bursty event if its number of relevant tweets in our dataset at least doubled in the future time window scope from the first detected time point. In these experiments, we define the **precision** and **recall** as follows:

$$\text{Precision} = \frac{a}{b} \tag{13}$$

$$\text{Recall} = \frac{a}{c} \tag{14}$$

where a is the number of the bursty events detected and they are also the real-world events, b is the number of the bursty events detected by the same algorithm, and c is the number of the bursty events detected by all the four algorithms and they are the real-world events.

DOI: 10.1002/cpe

6.4. Evaluation

In this section, we show the effectiveness evaluation of the three models, namely, PLSA, TwitterMonitor, and our BEE+. For each algorithm, we set the number of topics (clusters) K to 30 to detect the bursty events on December 19, 2014.

6.4.1. Effectiveness. Tables II and III show the comparison between these models in terms of the precision and recall of the top-K results. Overall, our BEE+ is the best in the **precision** for $K = 10$. For $K < 10$, the four models perform the same. The reason of the increased precision brought by our model is taking the hashtags into consideration. In the **recall**, the TM performs the best followed by our model for $K = 10$. We find that it is because the TM detected many bursty keywords for its grouping module, while our model ignores the unimportant bursty information, as same as our old BEE model [11].

6.4.2. Efficiency. Table IV shows the time each method needs for converging with different numbers of topic. Both our old BEE model and our BEE+ model are much faster than the two baselines. Because our model estimates the parameters incrementally, each parameter needs only some tight changes. While the two baselines begin with random initial value for their parameters each time.

6.4.3. Topic drifting. In this section, we illustrate the ability to track topic drifting of our model by case studying. We first initialize our BEE+ model then update its parameters every 10 min. Table V shows us the top event every 10 min from 10:00 AM to 10:30 AM on December 19, 2014. The events' top words are the words in the event-related posts having the most important weights in the sense

Table II. Precision at K for the four models.

Method	P@5	P@10	P@30
TM	0.800	0.500	0.400
PLSA	0.800	0.500	0.500
BEE	0.800	0.600	0.500
BEE+	**0.800**	**0.700**	**0.700**

TM, TwitterMonitor; PLSA, probabilistic latent semantic analysis; BEE, Bursty Event dEtection. The bold number is the best results.

Table III. Recall at K for the four models.

Method	P@5	P@10	P@30
TM	0.750	**0.800**	0.750
PLSA	0.750	0.733	0.625
BEE	0.875	0.733	0.733
BEE+	**0.875**	0.733	**0.800**

TM, TwitterMonitor; PLSA, probabilistic latent semantic analysis; BEE, Bursty Event dEtection. The bold number is the best results.

Table IV. Convergent time.

Method	P@5 (s)	P@10 (s)	P@30 (s)
TM	73	135	254
PLSA	85	141	296
BEE	25	42	76
BEE+	27	39	80

TM, TwitterMonitor; PLSA, probabilistic latent semantic analysis; BEE, Bursty Event dEtection.

DOI: 10.1002/cpe

Table V. Top event from 10 AM to 10:30 AM.

First detected time	Top words	Hash tag
10 : 00, December 19	一瞬间(one moment), 咬断(bit) 急刹(brakes), 米线(rice noodles) 送医(sent to hospital) 鲜血(blood)	6岁女孩车上吃米线 A 6-year-old girl bit her tongue when she was eating rice noodles on board
10 : 10, December 19	宣判(adjudge), 机长(Pilot in command) 车上(on board) 遇难 (victims) 女王(queen) 空难(crash) 获刑(be punished) 伊春(Yichun)	伊春空难机长 被判有期徒刑3年 Pilot in command of the “Yichun” Crash is punished.
10 : 20, December 19	芒果(MangGuo) 跨年(YearMix) 给出(give) 导演组(director) 意味着(means) 设计(design) 创意(creative) 信仰(belief)	在芒果跨年 YearMix part of MangGuo television station

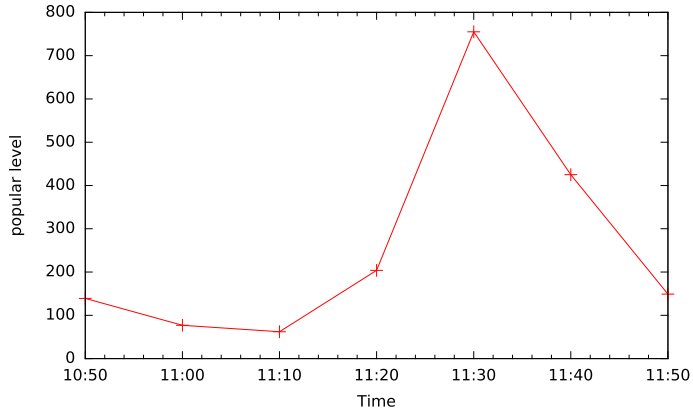

Figure 4. Topic drifting of the court session of ‘Yichun crash’.

of Term Frequency-Inverse Document Frequency (TFIDF) value, which is a classical evaluation for key words extraction. And the hashtags are the most frequent in those posts.

From the data of December 19, 2014, we extract an event of a sentence of 5 years on the Pilot in Command of ‘Yichun crash’. Figure 4 indicates the topic drifting of this event, popular level means how many posts talk about this event. The height represents how popular this event is. We can see that after 11 o’clock, this event exploits a lot because in real word, this event happened at 9 o’clock, and some news columns report it at 11 o’clock. And our algorithm detects it earlier than 11 o’clock.

Note that TM and PLSA cannot utilize the different temporal information of events to group keywords. From the results, we can see that the topically similar contents appear around the same time and our results also support our hypothesis. We can see that modeling the temporal information, BEE can depict burst characteristics well for analysis.

6.4.4. Efficiency. Our experimental environment is as follows.

1. Eight nodes: one master node and seven slave nodes
2. File system: Hadoop Distributed File System (HDFS)
3. Number of cores for each node: 32
4. Cache for each node: 128 MB
5. RAM for each node: 40 GB

In the experiment, our algorithm converges much faster than TM and PLSA models, which prove intuitively our algorithm’s efficiency. We use *scaleup*, *sizeup*, and *speedup* [19] and compare them to evaluate the efficiency of BEE+.

DOI: 10.1002/cpe

Sizeup. Sizeup measures how much longer the execution time is on a given system, when the dataset size is x times larger than the original dataset. Its definition is given by the following formula:

$$\text{Sizeup(data}, x) = \frac{T_x}{T_1} \tag{15}$$

where T_x is the execution time for processing an x times larger dataset and T_1 is the execution time for processing the original dataset.

Speedup. Speedup is defined as how much a distributed algorithm is faster than a corresponding sequential algorithm. It is defined by the following formula:

$$\text{Speedup} = \frac{T_1}{T_x} \tag{16}$$

where x is the number of machines, T_1 is the execution time of the algorithm with one machine, and T_x is the execution time of the parallel algorithm with x machines.

Scaleup. Scaleup refers to the ability of an m times larger system to perform an m times larger job in the same run time as the original system. The definition is as follows.

$$\text{Scaleup(data,}x) = \frac{T_1}{T_{xx}} \tag{17}$$

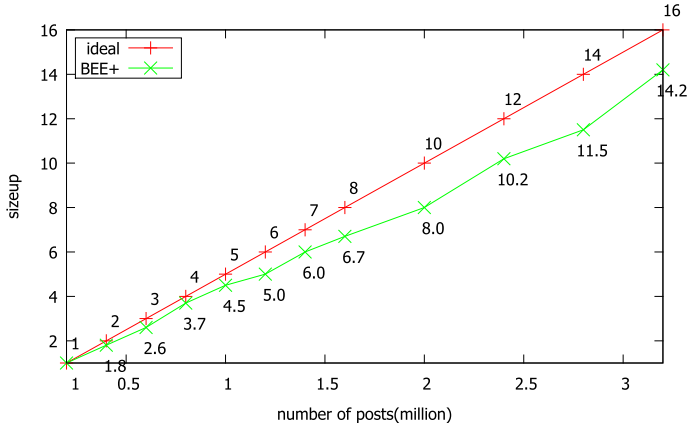

Figure 5. Sizeup performance evaluation. BEE+, Bursty Event dEtection.

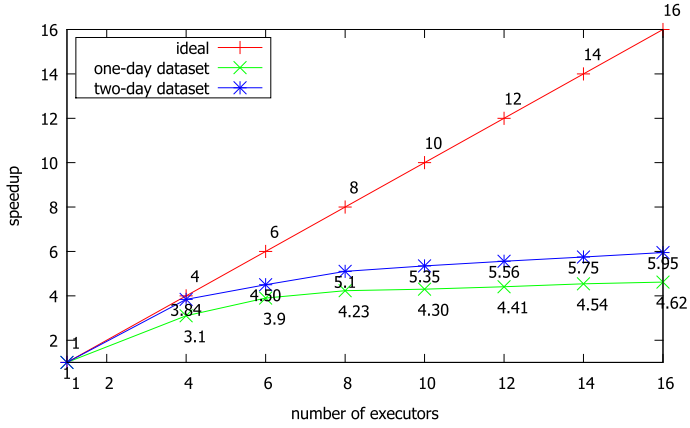

Figure 6. Speedup performance evaluation.

DOI: 10.1002/cpe

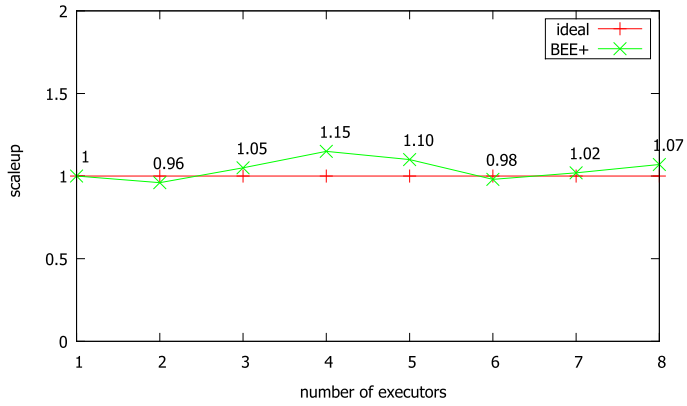

Figure 7. Scaleup performance evaluation. BEE+, Bursty Event dEtection.

where T_1 is the execution time for processing a certain data on one machine and T_{xx} is the execution time for processing x times data on x machines.

Figure 5 shows the sizeup result on our experimental environment, the result shows the **sublinear performance** for our BEE+. Increasing the size of the dataset simply makes the non-communication portion of the code and takes more time due to more I/O and more posts processing. Because I/O and CPU processing scale well with *sizeup*, we obtain sublinear performance.

From Figure 6, we could see that BEE+ achieves linear speedup when the computers are small. However, the improvement becomes gradually undramatic as the number of machines grows. This is expected because of both the increase in the absolute time spending in communication between machines and the increase in the fraction of the communication time in the entire execution time. When the fraction of the computation part dwindles, adding more machines cannot improve much speedup.

The scale-up performance of BEE+ is shown in Figure 7. We ran several different datasets on distributed machines from two executors to eight executors. Clearly, our method scales very well, it is able to keep the execution time almost *constant* as the dataset and machine sizes increase.

7. RELATED WORKS

Some event detection method focused in special event detection. Phuvipadawat and Murata [20] focused on detecting the breaking news from Twitter. The method of event detection and topic tracking in their approach is employed by clustering the similar tweets together. The similarity of two tweets of breaking news is measured by calculating the distance between their vector representation, which is a variant of the term frequency–inverse document frequency. But their algorithm only deals with the tweets with hashtag '#breakingnews'. Li *et al.* [21] proposed an event detection and analysis system to detect crime and disaster-related events from tweets. Their system supports to extract a geographic location for events and rank events based on the importance of events. Sakaki *et al.* [22] identified the bursty events about the earthquake that happened in Japan. The main method in their work is support vector machine, which was used to judge whether a tweet is about the earthquake or not. In summary, the aforementioned approaches are applicable to certain types of tweets (e.g., having a specific hashtag related to crime and disaster or containing the certain keywords). Some a priori knowledge about the events to be detected is needed. So, they cannot detect the general events for our problem without priori knowledge.

Several other people make a few efforts to detect the unspecific events without a priori knowledge. Mathioudakis and Koudas [4] group bursty keywords into events according to their co-occurrences in history tweets. Xie *et al.* [23] identify bursty co-occurrences between keywords quickly using dimension reduction, and detect the bursty events based on the co-occurrences by solving the optimization problem. Li *et al.* [24] split each tweet into non-overlapping segments (semantically meaningful information units). They clustered the bursty segments as event segments. However, the

DOI: 10.1002/cpe

aforementioned systems ignore the temporal information of events and cannot track the changes of events. So, the bursty importance of events and changes of events are not measured.

There are also some other event detection approaches that are based on the temporal information of the events. Weng and Lee [25] apply wavelet transformation to fit the temporal information of each word. The events are formed by using modularity-based graph partitioning algorithm. However, the similarity between words measured by the cross correlation between signals cannot differentiate the busty distinct events that happened at the same period by coincidence. Diao *et al*. [1] proposed an LDA-based topic model that exploits this idea to find the bursty global events. However, their model is not immediately applicable for our problem. Firstly, it is infeasible to model the interest of all the users, because most of the users release very little tweets. Secondly, their model detects the bursty events offline.

Our BEE+ can distinguish the events that happened around the same period well by modeling the temporal information of events with the observation that the topical similar contents appear around the same time. What is more, our algorithm is able to track the topic drifting.

8. CONCLUSION

To extract useful real-time information from the massive microblogs' data stream, in this paper, we proposed an incremental temporal topic model (BEE+) for detecting bursty events. Based on our BEE model [11], we take the hashtag in microblogs into consideration, which increases the precision in BEE+. In order to achieve real time, a distributed execution framework is designed based on Spark engine, which is scalable for big data processing with low latency. We conduct experiments using the Weibo's posts of December 19, 2014. Comparing with PLSA and TM, the results show that our algorithm can outperform those two existing models for detecting the meaningful bursty events. We conduct a case study to illustrate the efficiency of BEE+ to track the topic drifting.

For further work, we plan to look into methods that fuse multiple data resources to improve BEE+'s performance on detecting bursty event. Besides, we plan to transform our parameter-parallel method to model parallel, which is more efficient, in the future.

ACKNOWLEDGEMENT

This work is supported by NSFC Program (No. 61472022), China MOST project (No.2012BAH 46B04), and SKLSDE-2014ZX-04.

REFERENCES

1. Diao Q, Jiang J, Zhu F, Lim EP. Finding bursty topics from microblogs. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*: Association for Computational Linguistics, 2012; 536–544.
2. Lee P, Lakshmanan LVS, Milios EE. Incremental cluster evolution tracking from highly dynamic network data. *2014 IEEE 30th International Conference on Data Engineering (ICDE),* IEEE, 2014; 3–14.
3. Yan X, Guo J, Lan Y, Cheng X. A biterm topic model for short texts. *Proceedings of the 22nd International Conference on World Wide Web*: International World Wide Web Conferences Steering Committee, 2013; 1445–1456.
4. Mathioudakis M, Koudas N. Twittermonitor: trend detection over the twitter stream. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data,* ACM, 2010; 1155–1158.
5. Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science (JAsIs)* 1990; **41**(6):391–407.
6. Hofmann T. Probabilistic latent semantic indexing. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* ACM, 1999; 50–57.
7. Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. *Journal of the Machine Learning Research* 2003; **3**: 993–1022.
8. Boyd-Graber JL, Blei DM. Syntactic topic models. *Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems (NIPS 2008)*, Vancouver, British Columbia, Canada, 2008; 185–192.
9. Wang X, McCallum A. Topics over time: a non-Markov continuous-time model of topical trends. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* ACM, 2006; 424–433.
10. Hong L, Davison BD. Empirical study of topic modeling in twitter. *Proceedings of the First Workshop on Social Media Analytics,* ACM, 2010; 80–88.

DOI: 10.1002/cpe

11. Li J, Tai Z, Zhang R, Yu W, Liu L. Online bursty event detection from microblog. In *the 7th IEEE/ACM International Conference on Utility and Cloud Computing,* ACM, 2014.
12. Gruber A, Weiss Y, Rosen-Zvi M. Hidden topic Markov models. *International Conference on Artificial Intelligence and Statistics*, 2007; 163–170.
13. Zhao WX, Jiang J, Weng J, He J, Lim EP, Yan H, Li X. Comparing twitter and traditional media using topic models. In *Advances in Information Retrieval*. Springer, 2011; 338–349.
14. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation,* USENIX Association, 2012; 2–2.
15. Marcus A, Bernstein MS, Badar O, Karger DR, Madden S, Miller RC, Arenz O. Twitinfo: aggregating and visualizing microblogs for event exploration. *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems,* ACM, 2011; 227–236.
16. Wang X, Zhai CX, Hu X, Sproat R. Mining correlated bursty topic patterns from coordinated text streams. *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* ACM, 2007; 784–793.
17. Marcus A, Bernstein MS, Badar O, Karger DR, Madden S, Miller RC. Twitinfo: aggregating and visualizing microblogs for event exploration. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* ACM, 2011; 227–236.
18. Chen Y, Amiri H, Li Z, Chua TS. Emerging topic detection for organizations from microblogs. *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval,* ACM, 2013; 43–52.
19. Li N, Li Z, He Q, Shi Z. Parallel implementation of apriori algorithm based on mapreduce. *Proceedings of 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2012; 236–241.
20. Phuvipadawat S, Murata T. Breaking news detection and tracking in twitter. *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Vol. 3. IEEE, 2010; 120–123.
21. Li R, Lei KH, Khadiwala R, Chang KC. Tedas: a twitter-based event detection and analysis system. *2012 IEEE 28th International Conference on Data Engineering (ICDE),* IEEE, 2012; 1273–1276.
22. Sakaki T, Okazaki M, Matsuo Y. Earthquake shakes twitter users: real-time event detection by social sensors. *Proceedings of the 19th International Conference on World Wide Web,* ACM, 2010; 851–860.
23. Xie W, Zhu F, Jiang J, Lim EP, Wang K. Topicsketch: real-time bursty topic detection from twitter.
24. Li C, Sun A, Datta A. Twevent: segment-based event detection from tweets. *Proceedings of the 21st ACM International Conference on Information and Knowledge Management,* ACM, 2012; 155–164.
25. Weng J, Lee BS. Event detection in twitter. *ICWSM*, 2011.

DOI: 10.1002/cpe