Applied Soft Computing 70 (2018) 1010–1023

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

An effective hot topic detection method for microblog on spark

Wei Ai[a,*], Kenli Li[a], Keqin Li[a,b]

[a] School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 26 September 2016
Received in revised form 25 August 2017
Accepted 28 August 2017
Available online 7 October 2017

Keywords:
Big data
Hot topic detection
Microblog
Non-iterative clustering algorithm
Spark

ABSTRACT

With the emergence of the big data age, methods for quickly and accurately obtaining valuable hot topics from the vast amount of digitized textual material have attracted much attention. In this work, we focus on topic detection in microblogs in the big data environment. Different from existing approaches, we solve this problem in a distributed way. Specifically, we propose a non-iterative algorithm called parallel two-phase mic-mac hot topic detection (TMHTD), and implement it in the Apache Spark environment. The proposed TMHTD method includes two phases, i.e., the micro-clustering phase and the macro-clustering phase. To improve the accuracy of hot topic detection, three optimization methods, along with TMHTD, are proposed. To handle large databases, we deliberately design a group of MapReduce jobs to concretely accomplish hot topic detection in a highly scalable way. We compare the TMHTD algorithm with the general single-pass algorithm and the Latent Dirichlet Allocation (LDA) algorithm. Our experiments are carried out on real-life data sets gathered from the Sina Weibo API. Extensive experimental results indicate that the accuracy and performance of the TMHTD algorithm are significant improvements over previous methods. More specifically, the *F*-measure value of the TMHTD algorithm shows a 6% and 8% improvement over the general single-pass algorithm and the LDA algorithm, respectively. The run time of the TMHTD algorithm is 7 times and twice as superior to the general single-pass algorithm and the LDA algorithm, respectively.

1. Introduction

1.1. Motivation

With the advent of the big data era, the amount of data available has increased on a large scale in various fields. Every day, 2.5 quintillion bytes of data are created, and 90 percent of the data in the world today has been produced within the past two years [1]. The vast amount of digitized textual material is now available on the internet, but high-speed connectivity and the explosion in the volume of digitized textual content available online have given rise to information overload [2]. Clearly, although there are large amounts of worthwhile information contained in the huge quantities of data, it is impossible for people to absorb all pertinent information because humans have a limited capacity to assimilate information. Therefore, these characteristics make it extremely challenging to explore the large volumes of data and extract useful information or knowledge quickly and accurately from big data with data-mining methods [3].

Topic detection (TD) has received considerable attention and has been widely accepted as a promising research issue in data-mining to meet this challenge. TD can be seen as a clustering of events that helps analysts separate the wheat from the chaff in massive textual materials. By exploring and organizing the content of textual materials, TD attempts to identify topics that will enable us to aggregate disparate pieces of information into manageable clusters automatically. At the same time, hot topic detection provides a way to effectively track hot events and focus on important topics.

Compared with the traditional topic detection applications, such as online news and newspapers, microblogs have become a public social media with quick, short, a large number of participants, easy characters, high impact and fast information spread [4,5]. At the same time, since microblogs are very sensitive to hot topics, microblogs play an important role in the spread of hot events. Due to the comments, reports and forwards of a large number of users, hot events have been discussed extensively on microblogs before the responses of traditional media. As a result, microblogs have become one of the main platforms for people to access and publish information, and hundreds of millions of microblogs are released per minute.

* Corresponding author.
E-mail addresses: aiwei@hnu.edu.cn (W. Ai), lkl510@263.net (K. Li), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.asoc.2017.08.053
1568-4946/

Hot topic detection for microblogs under big data environment not only provides a stronger and more flexible response to various requests in social and business applications, but also has far-reaching social and economic values. Users can grasp social hot spots, follow the tendencies of society in real-time, and keep up with significant events. Enterprises can find opportunities among hot topics that attract users, be informed of the developments in related fields, and collect information about the users' mental activities. Professional media and authorities can keep abreast of the directions of public opinions and current hot social events on many issues, so that they can accordingly implement policy and effectively guide public opinions. As a result, hot topic detection in microblogs for big data has aroused great concern in industry and academia.

Similar to many big data applications, hot topic detection in microblogs is also affected by the characteristics of big data seriously, and there are also many problems in practice. (1) The enormous amount of microblog data can be regarded as hundreds of millions of various input variables, but some of them contains very little information. In addition, only part of microblogs contain popular events information, most of microblogs are about people's daily lives. Therefore, it is necessary to extract the information that can most properly describe the event and eliminate the noise data. (2) Since topic detection in microblogs is a timely task and the amount of data is huge, it is difficult to use common algorithms to deal with. (3) Due to competitive business pressures and user demands, the requirement for real time hot topic detection processing and the detection validity becomes higher than ever before [6,7]. Therefore, how to accurately and quickly obtain valuable hot topic information from big data environments is an important research problem. This research not only focuses on the efficiency of the hot topic detection methods but also maintains acceptable accuracy.

1.2. Our contributions

Motivated by the above observations, we specially focus on hot topic detection for microblog under big data environment in this paper. A distributed two-phase mac-mic hot topic detection (TMHTD) approach is proposed, and implemented on the Apache Spark. To improve the detection accuracy of hot topic, together with TMHTD, three optimization methods are also proposed. To handle large-scale data, a set of MapReduce operations are designed to achieve hot topic detection. A large number of experimental results show that the performance and accuracy of the TMHTD algorithm are significantly improved over previous methods.

In summary, our contributions are as follows.

- We propose an effective method called TMHTD to detect hot topics in microblogs in a large dataset, which includes the following two phases: the macro-clustering phase and micro-clustering phase.
- We design three optimization methods for TMHTD to effectively improve the detection accuracy of hot topic. The optimization methods include the following aspects: text selection for micro-clustering, topic selection for macro-clustering, and construction of a coarse/fine-grained similarity computation method for the single-pass clustering algorithm.
- We import the TMHTD method and the related algorithms into the Spark cloud computing environment, and a group of MapReduce jobs are deliberately designed and coordinated to accomplish the concrete computation of TMHTD.
- We evaluate the performance of the proposed solution through extensive experiments under different data set sizes and varying Spark platform configurations. A large number of experimental results show that the performance and accuracy of the TMHTD algorithm are significant improvements over previous methods. More specifically, we compare the TMHTD algorithm with the general single-pass and the LDA algorithms. The F-measure value of the TMHTD algorithm shows an improvement of up to 6% and 8% compared with the general single-pass and LDA algorithms, respectively. The run time of the TMHTD algorithm is 7 times and twice as superior to the general single-pass and LDA algorithms, respectively.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 presents useful preliminaries. Section 4 describes the TMHTD method and the related algorithms. Section 5 imports the algorithms into the Spark cloud computing environment and implements the parallelized TMHTD. Section 6 evaluates the proposed algorithms with experiments. Section 7 concludes the paper.

2. Related work

In the past, many research groups attempted to study the topic detection [2,8–13]. In [2], the authors put forward a hot topic detection method based on multidimensional sentence modeling and timeline analysis. In [10], the authors proposed a three-level hierarchical Bayesian model named latent Dirichlet allocation (LDA), which is a generative probabilistic model for collections of discrete data such as text corpora. In [8], in order to improve the detection of relevant documents on the basis of terms found in queries, the authors proposed an approach to automatic indexing and retrieval, which takes advantage of implicit higher-order structure in the association of terms with documents. In [9], based on a statistical latent class model for factor analysis of count data, the authors proposed a novel approach to automated document indexing. In [12], the authors presented an online topic model called OLDA that identifies new topics of text streams and changes in these new topics over time, and automatically captures the thematic patterns. In [13], the authors presented a regularized topic model which enhances topic learning by using word co-occurrence information statistics. In [11], the authors presented a topic detection model named the temporal discriminative probabilistic model (DPM), which is theoretically equivalent to a classical vector space model with temporally discriminative and weights feature selection. All of the above studies have been successful applied to deal with normal texts (e.g., academic papers and news articles) without considering the particularity of microblog.

According to the characteristics of microblog, most of the existing topic detection methods are designed for microblog [14–16]. In [17], the authors proposed an improved single-pass clustering technique that uses the potential dirichlet allocation (LDA) model instead of traditional vector space model, which can extract hidden microblog topic information. In [18], the authors proposed an incremental clustering framework that can quickly detect hot emerging topics based on temporal features and a range of contents. In [19], the authors adopted a topic detection model of microblog named topic-coreterms latent dirichlet allocation (TC-LDA) model. In [20], the authors developed a novel biterm topic model (BTM) which can capture topics within short texts by explicitly modeling word co-occurrence patterns throughout corpus. In [21], the authors developed an efficient hot topic detection algorithm that can handle a large amount of tweets online. In [22], the authors proposed a parallel two-phase mic-mac hot topic detection(TMHTD) method specially designed for microblogs. However, the above works cannot effectively deal with hot topic detection of microblog when facing massive data. Unlike these methods, we use a distributed approach to solve hot topic detection.

Apache Spark was originally presented as a distributed framework [23], which is a fast and general engine for parallel processing, and has been applied in many fields. Lin et al. presented an unified cloud platform for batch log data analysis with the combination of Hadoop and Spark [24]. Zhou et al. proposed a novel MapReduce based $N-1$ analysis algorithm to address the static security analysis problem by using Spark cloud computing platform [25]. Solaimani et al. developed a statistical technique for online anomaly detection, which uses Spark over heterogeneous data from multi-source VMware performance data [26]. Privitera et al. proposed a solution for soft real-time GPRS traffic analytic using Spark [27]. Koliopoulos et al. discussed a distributed framework for Weka which maintains its existing user interface for big data mining using Spark [28]. Zhang et al. proposed an efficient distributed frequent itemset mining algorithm (DFIMA) which has been implemented using Spark to further improve the efficiency of iterative computation [29]. Liang et al. presented a LU decomposition based block-recursive algorithm for large-scale matrix inversion on the Spark parallel computing platform [30]. Liu et al. proposed a referential architecture on the platform solution to overcome the problems in tremendous of healthcare big data process based on Spark [31].

Spark provides another choice for large-scale data processing. In [23], the authors proposed a new framework called Spark not only have been successfully applied in implementing large-scale data intensive applications on clusters of unreliable machines, but also can be used to interactively query a 39 GB dataset with sub-second response time. In [32], the authors provided the open source computing framework unifies streaming, batch, and interactive big data workloads to unlock new applications. In [33], the authors presented an overview and brief tutorial of deep learning in mobile big data analysis and discusses a scalable learning framework over Apache Spark. In [33], the authors presented Apache spark as a unified cluster computing platform, which is very useful for executing and storing big data analysis of smart grid data. In [34], the authors aims to bring some justice by directly evaluating the performances of Spark and Flink. In [35], the authors identified several meaningful information from several sample big data sources, and explored the concept of big data analysis.

3. Preliminaries

3.1. Problem definition

Denote D as a corpus documents set over T time intervals, i.e., $D=\{D_1, D_2, \ldots, D_T\}$, $D_i \cap D_j = \emptyset$ for all $1 \le i \ne j \le T$. In our work, we simplify a document into a bag of feature terms or keywords. An object $d_i \in D_t$ can be expressed as $d_i = \{f_{i,1}, f_{i,2}, f_{i,3}, \ldots, f_{i,k}\}$ where $d_i.f_{i,k}$ is a feature item extracted from document d_i by applying the typical pre-processing method. A topic top_i is a seminal activity or event, in addition to all directly related activities and events. A hot topic is defined as a topic that appears frequently over a period of time [36]. The problem of hot topic detection can be defined as follows.

Definition 3.1 *(Hot topic detection). Hot topic detection* aims to group thematically related documents from a temporal document set into an unknown number of topics and then find a set of topics that appear frequently over a period of time.

Assuming that there is a document set C, a hot topic detection approach divides C into K disjoint clusters $TOP=\{top_1, top_2, \ldots, top_K\}$. We use *Heat(TOP)* to denote the frequency vector of the topic set *TOP* contained in the document C. Then, the frequency vector of the topic set *TOP* can be written as

$$Heat(TOP) = \{Heat(top_1), Heat(top_2), \ldots, Heat(top_K)\}. \quad (1)$$

Finally, all topics are sorted in descending order by frequency, and the top k $(1 \le k \le K)$ topics are selected as the final hot topics.

Note that we consider the application to microblog. A critical problem is how to detect and extract hot topics from a large amount of microblog document sets quickly and accurately. Thus, the objective of this paper is to improve the accuracy and efficiency of hot topic detection in microblogs for big data applications.

3.2. The general framework

In this section, we present a general framework for processing hot topic detection following this strategy. The main steps of hot topic detection are depicted in Fig. 1 and are described in detail as follows.

Step 1. Text preprocessing. Preprocessing microblog text is the first step in hot topic detection. The microblog information needs to be structured in such a way that the computer can recognize it. Text preprocessing includes word segmentation and filtering of meaningless words. According to the language we need to process, we must choose the relevant word segmentation tool, such as ICTCLAS for a Chinese corpus. After the word segmentation, we remove meaningless words, such as stop words and characters not in Latin or Chinese. We obtain a set of words (i.e., feature items) after preprocessing the microblog texts.

Step 2. Feature selection and weighting. Feature items from different parts of the text contribute differently to the main idea of the microblog text. Nouns, verbs, adjective, numeral and name entities (NEs) are usually selected as the features to represent each document in general microblog hot topic detection.

After feature selection, the feature weight values should be calculated. There are two main methods to calculate feature weights as follows: Boolean weights and TF-IDF (term frequency-inverse document frequency) weights. The TF-IDF measure is widely used to weight the features [37].

Step 3. Text model construction. Usually, the vector space model (VSM) [38] is used to express microblog text. The basic idea of the vector space model is to represent text by using space vectors, where each dimension of the vector represents a feature item and the value of each dimension represents the weight of the corresponding feature item. Then, text d_i can be converted to a vector. For example, for text d_i, the vector space model of d_i is $V(d_i)$, $V(d_i) = (f_{i,1}, w_{i,1}; f_{i,2}, w_{i,2}; \ldots; f_{i,K}, w_{i,K})$, where $f_{i,k}$ is the kth feature item and $w_{i,k}$ is the weight of $f_{i,k}$.

Step 4. Topic clustering. Many documents are clustered using the single pass clustering algorithm [39], which is a technique that has been proven to be reasonably effective for TD. Single-pass is a non-iterative clustering algorithm, which has an obvious advantage for processing massive data. However, it has low accuracy when applied to microblog hot topic detection because it is sensitive to the order of the inputs, and it takes a long time to process small clusters with few microblogs because there are still too many topic-unrelated noise microblogs.

3.3. Apache Spark

Apache Spark is one of most famous cloud computing frameworks for large-scale data processing. It can work in the standalone cluster mode or on top of the next-generation Hadoop cluster called YARN. It provides in-memory cluster computing, which is 100 times faster than Hadoop MapReduce [40]. Similar to Hadoop, Spark also possesses good characteristics, such as high scalability and fault tolerance.

The core of the programming model features resilient distributed data sets (RDD), which is a new distributed memory abstraction. RDD represents a collection of distributed items that

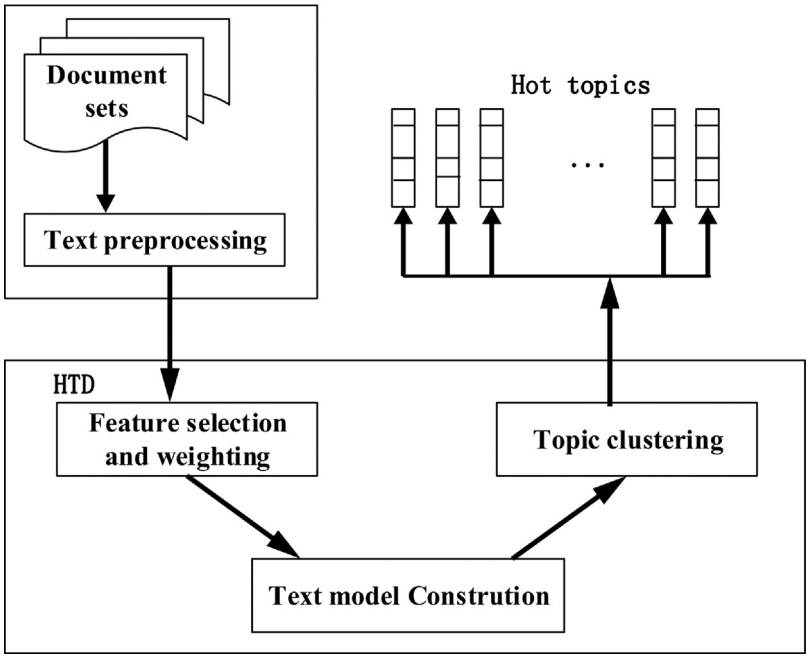

Fig. 1. The general framework.

can be manipulated across many computing nodes concurrently. It allows us to store a data cache in memory and perform computations for the same data directly from memory. Therefore, the Spark platform saves large amounts of disk I/O operation time. After the RDD is constructed, the programmer can perform transformations and actions. Transformations (e.g., map and filter) create new data sets from the existing RDD or the input file, while actions (e.g., reduce, collect, count, length, and saveastextfile) return a value after executing calculations on the RDD [41].

4. Two-phase mic-mac hot topic detection method

The inadequacy of the traditional single-pass cluster algorithm for hot topic detection exists because (1) it is very time consuming for processing massive data and (2) it is limited in accuracy. To improve the time efficiency of the single-pass clustering algorithm, we propose a parallel two-phase mic-mac hot topic detection (TMHTD) method based on an improved single-pass clustering algorithm, and import the TMHTD method and the related algorithms into the Spark cloud computing environment. To improve the accuracy of hot topic detection, we design three optimization methods for TMHTD, which include the following aspects: text selection for micro-clustering, topic selection for macro-clustering, and construction of a fine/coarse-grained similarity computation method for the single-pass clustering algorithm.

4.1. Sketch of two-phase mic-mac hot topic detection method

Definition 4.1 *(Two-phase Mic-mac Hot Topic Detection method, TMHTD).* In this paper, we propose a two-phase mic-mac hot topic detection (TMHTD) method, which consists of two phases: the micro-clustering phase and the macro-clustering phase. In the first phase, original data sets are partitioned into a group of smaller data subsets by using TMHTD algorithm. Then, these data subsets are clustered into many topics by using an improved non-iterative single-pass cluster algorithm. The main purpose of the first phase is to implement the improved single-pass algorithm on Spark, and improve the run time and accuracy of the single-pass algorithm. In the second phase, these topics are integrated into a single data set, and then the data set is clustered again according to an improved non-iterative single-pass clustering algorithm to obtain the final detection results. The main purpose of the second phase is to effectively detect hot topics.

Fig. 2 illustrates the workflow of our advanced solution, which receives a large dataset as the input and produces hot topics after the application of the TMHTD method. In the beginning, the text selection method reduces the number of document sets from M to m ($m < M$), and topic-unrelated text is removed. The details of the text selection process will be discussed in Section 4.2.

In the first phase, the document sets are clustered into many small topics using an optimal single-pass algorithm, producing intermediate results. Before the second phase, we filter out low frequency topics and reduce the number of micro-topics in order to improve the accuracy of the clustering result. The details of the topic selection process will be discussed in Section 4.3. In the second phase, the micro-topics are further clustered into macro-topics using an optimal single-pass algorithm to obtain the final hot topic sets.

The following two similarity computation methods are introduced in the optimal single-pass algorithm: a fine-grained similarity computation method and a coarse-grained similarity computation method. The fine-grained similarity computation method is used for the micro-clustering process, while the coarse-grained similarity computation method is used for the macro-clustering process. The details of these two methods will be discussed in Section 4.4.

The two phases of our approach, which are based on the two levels of topics for the big data environment, are implemented in Spark. The TMHTD algorithm is essentially a parallel algorithm, which includes the following two phases: the micro-clustering phase and macro-clustering phase. Before the first phase, an optimization method (text selection for micro-clustering) is adopted for TMHTD to improve the accuracy of hot topic detection and produce intermediate results. The intermediate results are then clustered in two phases, and the final set of hot topics is obtained. If the Spark platform is used, these intermediate results generated in the first phase only need to be stored in memory, without having to be stored in HDFS, greatly improving the performance of the algo-

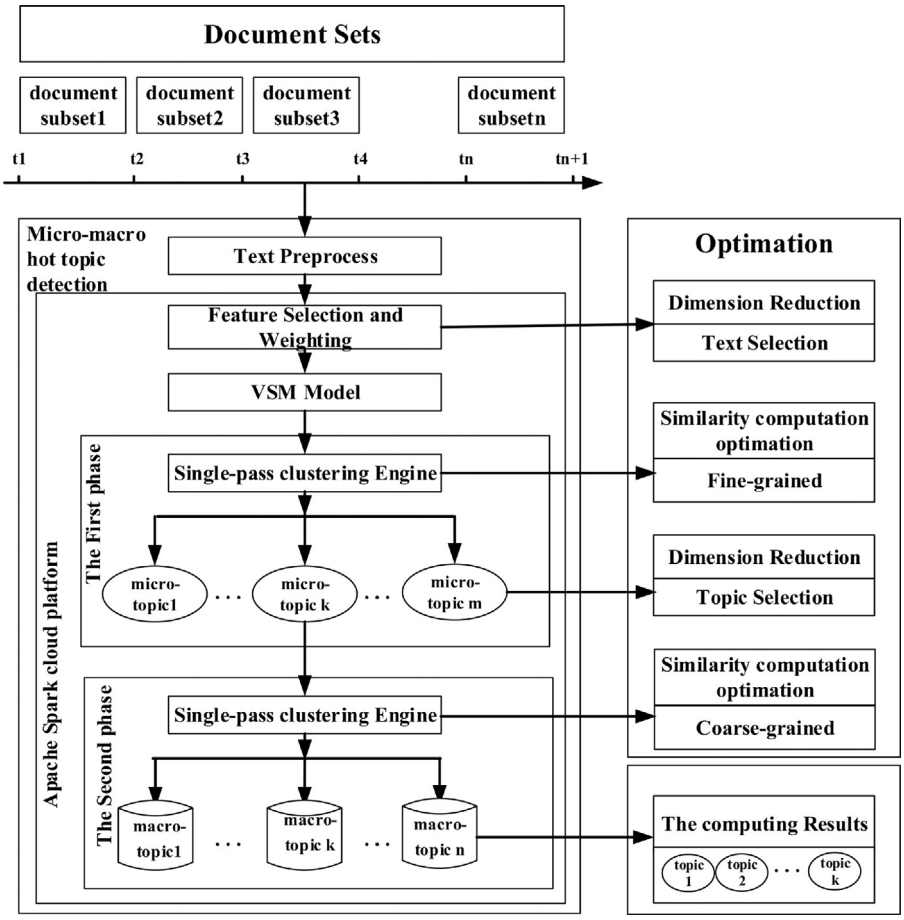

Fig. 2. Flow of TMHTD approach.

rithm. TMHTD is implemented based on the MapReduce framework and Spark platform, and the details of the parallel process will be discussed in Section 5.

4.2. Text selection for micro-clustering

There are a lot of topic-unrelated texts in data sets. Such texts not only influence the efficiency of the clustering method, but also lead to significant noise. Therefore, these topic-unrelated texts should be removed as much as possible based on the contribution of their feature items to the topic.

Given a document set D, after preprocessing, the text $d_i \in D$ is expressed as the collection of feature items: $d_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,K}\}$. Then, we can build a feature vector space of every text as follows: $V(d_i) = (f_{i,1}, w_{i,1}; f_{i,2}, w_{i,2}; \ldots; f_{i,K}, w_{i,K})$. The weight of every feature item can be calculated using the normalized TF-IDF function. Therefore, we have:

$$w_{i,k} = \frac{ff_{i,k} \times \log(N/n_{i,k} + 0.01)}{\sqrt{\sum_{j=1}^{M} \left[ff_{i,k} \times \log(N/n_{i,k} + 0.01) \right]^2}}, \tag{2}$$

where N is the number of texts, $n_{i,k}$ is the number of texts that have feature item $f_{i,k}$, and $ff_{i,k}$ is the frequency of feature item $f_{i,k}$ in text d_i.

Accordingly, our strategy for determining the contributions of feature items considers the following factors: feature frequency and feature bursty. To effectively measure the contributions of feature variables, we introduce a score function to determine whether the text is related to the topic. Before defining the score function, we first introduce the concepts of frequency and bursty of a feature.

Definition 4.2 *(Frequency of the feature:).* The factor is used to evaluate how many texts contain feature $f_{i,k}$. The frequency of the feature is defined as

$$fr(f_{i,k}) = \frac{N(f_{i,k})}{Num}, \tag{3}$$

where Num is the total number of documents, and $N(f_{i,k})$ is the number of documents containing feature $f_{i,k}$.

$$I(f_{i,k}) = \begin{cases} fr(f_{i,k}), & \text{if } \theta_2 < fr(f_{i,k}) < \theta_1; \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Definition 4.3 *(Bursty of the feature:).* The factor is used to evaluate the document frequency over a finite time. A word is bursty if it appears in a large number of documents and exhibits high document frequency over a finite time. The bursty of the feature is defined as

$$\begin{aligned} Burst(f_{i,k}) &= \max_{t \in (0,T)} \left\{ \frac{Freq_t(f_{i,k})}{\bar{Freq}_{t-1}(f_{i,k})} \right\} \\ &= \max_{t \in (0,T)} \left\{ \frac{Freq_t(f_{i,k}) \times (n-1)}{\sum_{r=1}^{t-1} Freq_r(f_{i,k})} \right\}, \end{aligned} \tag{5}$$

where $Freq_t(f_{i,k})$ is the number of documents containing feature $f_{i,k}$ at time t, and $\bar{Freq}_{t-1}(f_{i,k})$ is the average number of documents containing feature $f_{i,k}$ before time t.

$$CF(f_{i,k}) = I(f_{i,k}) \times Burst(f_{i,k}). \tag{6}$$

According to Eq. (6), a large $CF(f_{i,k})$ means $f_{i,k}$ has a large contribution in d_i. In this paper, a keyword is regarded as a feature item with a

high value of the contribution function. We select the top m feature items, and transform them into a corresponding keyword set KY according to the values of the energy function.

$$Score(d_i) = \sum_{k=1}^{K} w_{i,k} \times CF(f_{i,k}), \tag{7}$$

where $w_{i,k}$ represents the weight of the feature $f_{i,k}$ in the text d_i, and the contribution function $CF(f_{i,k})$ returns the contribution value associated with $f_{i,k}$. Thus, this score formula evaluates the topic-related likelihood of text d_i by considering the frequency and bursty of each feature in text d_i.

Algorithm 1. Text Selection for Micro-clustering Algorithm (Micro-TSA)

Require:
D: The document set;
δ: The threshold for score.
Ensure:
DD: The document set that reduce noise text;
KY: A set of m keyword feature variables.
for each $d_i \in D$ **do**
  Receive the number K of feature items from d_i
  $k \longleftarrow 1$
  while $(k \neq K)$ **do**
    Compute the weight value $w_{i,k}$ by Eq. (2)
    Compute the contribution value $CF(t_{i,k})$ by Eq. (6)
    $k \longleftarrow k+1$
  end while
  Compute the score value $Score(d_i)$ by Eq. (7)
  Sort feature items in descending order by $CF(t_{i,k})$
  Put top m item variables to $key_i[0, \ldots, m-1]$
  $KY \longleftarrow Key_i[0, \ldots, m-1]$
  if $Score(d_i) < \delta_1$ **then**
    Remove d_i from D
  end if
end for
Set $DD \longleftarrow D$
return DD, KY

4.3. Topic selection for macro-clustering

In the topic clustering process, the micro-topics are clustered again into corresponding macro-topics by the single-pass algorithm. The heat value of the topic is used to measure the topic frequency in the clustering process [42]. There are some low frequency topics in the micro-topic set. The heat values of the low frequency topics are much smaller than those of the hot topics. To improve the accuracy of the clustering result, we filter the low frequency topics and reduce the number of micro-topics before the micro-topics are further clustered into corresponding macro-topics. The topic selection method based on the heat of the topic is as follows:

First, let $C = \{top_1, top_2, \ldots, top_M\}$ be a micro-topic set, where M is the number of topics. Let $Heat(top_i)$ denote the heat of top_i. Then, $Heat(top_i)$ can be expressed as

$$Heat(top_i) = PN(top_i) + CN(top_i) + RN(top_i) + MN(top_i), \tag{8}$$

where $PN(top_i)$ is the number of participators of top_i, $CN(top_i)$ and $RN(top_i)$ are the number of comments and retweets of top_i, and $MN(top_i)$ is the microblog number contained in top_i.

Second, we sort the topics by heat values in descending order. Fig. 3 shows an example of $Heat(top_i)$, with the heat values in descending order. From the results, we can observe that the value of $Heat(top_i)$ has a sudden change when the topic is low frequency. The jump point of the topic heat is defined as the sudden change point of the topic heat compared to the total topic heat, and the heat

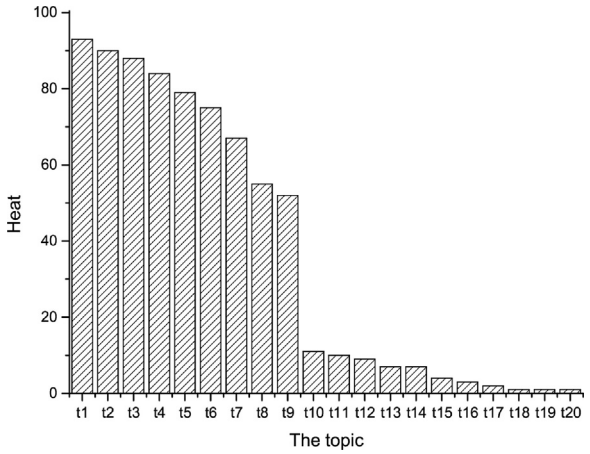

Fig. 3. An example of the micro-topics' heat.

change rate is used to find the jump point. Let $Rate(top_i)$ denote the heat change rate of top_i. Then, $Rate(top_i)$ is calculated by

$$Rate(top_i) = \left| \frac{Heat(top_{i+1}) - Heat(top_i)}{Heat(top_i)} - \frac{Heat(top_i) - Heat(top_{i-1})}{Heat(top_i)} \right|. \tag{9}$$

Last, based on the heat change rates of the topics, further filtering is conducted. Given a threshold δ, if $Rate(top_i) < \delta$, the topics for which the heat value is smaller than or equal to the value of the jump point will be filtered. Otherwise, top_i will be retained as a high frequency micro-topic variable for further clustering. Thus, the number of dimensions of the topic set is reduced from M to m $(m < M)$.

The detailed steps of the topic selection for the macro-topic clustering process are illustrated in Algorithm 2.

Algorithm 2. Top Selection for Macro-clustering Algorithm (Macro-TSA)

Require:
C: The micro-topic set;
δ: The threshold for the jump point.
Ensure:
TS: A set of m important mircro-topics variable of C.
Create an empty string array TS
for each $top_i \in C$ **do**
  Calculate $Heat(top_i)$ for target micro-topic variable top_i by Eq. (8)
end for
Sort the micro-topic variables top_i in descending order by $Heat(top_i)$
for $i = 2$ to n **do**
  Calculate $Rate(top_i)$ by Eq. (9)
  if $Rate(top_i) > \delta$ **then**
    $m \longleftarrow i$
    break
  else
    $i \longleftarrow i+1$
  end if
end for
if $m \neq 0$ **then**
  Put m micro-topic variable to $TS[0, \ldots, m]$
end if
return TS

4.4. Fine/coarse-grained similarity computation of single-pass algorithm

In this paper, the single-pass clustering technique is used to group the document sets into clusters. Cosine similarity is used as

a measurement to calculate the similarity between text d_i and d_j, namely,

$$\begin{aligned} sim(d_i, d_j) &= \cos(\vec{d}_i, \vec{d}_j) \\ &= \frac{\vec{W}_i \cdot \vec{W}_j}{\left\|\vec{W}_i\right\|_2 \times \left\|\vec{W}_j\right\|_2} \\ &= \frac{\sum_{k=1}^{n} \vec{W}_{i,k} \times \vec{W}_{j,k}}{\sqrt{\sum_{k=1}^{n} \vec{W}_{i,k}^2} \sqrt{\sum_{k=1}^{n} \vec{W}_{j,k}^2}}, \end{aligned} \tag{10}$$

where n is the total number of feature items in the two feature vector, $\vec{W}_i$ and $\vec{W}_j$ are the weight vectors of texts d_i and d_j, respectively. $\vec{W}_{i,k}$ and $\vec{W}_{j,k}$ represent the weights of the kth feature items of texts d_i and d_j, respectively.

In the TMHTD method, we introduce two similarity computation methods: a fine-grained similarity computation method and a coarse-grained similarity computation method. The fine-grained similarity computation method is used when the single-pass algorithm is applied to the micro-clustering process, while the coarse-grained similarity computation method is used when the single-pass algorithm is applied to the macro-clustering process. The traditional one-vector model is adopted by the original single-pass algorithm, under such conditions, the accuracy of the similarity computation is reduced if the vector contains some noisy feature items.

4.4.1. Fine-grained similarity computation

The old calculation was improved by us, and a three-dimensional vector model was proposed by us. The three-dimensional vector model not only considers more diverse criteria but also uses an additional factor associated with the term position in each text to identify the text similarity:

$$\cos(\vec{d}_i, \vec{d}_j) = \begin{cases} \alpha \times \cos(KWV_i, KWV_j), & \text{if a feature term in keyword list;} \\ \beta \times \cos(FSV_i, FSV_j), & \text{if a feature term in first sentence;} \\ \gamma \times \cos(OPV_i, OPV_j), & \text{if a feature term in the other part;} \end{cases} \tag{11}$$

where

- $\cos(vector_i, vector_j)$ is the cosine similarity of $vector_i$ and $vector_j$;
- α, β, and γ are the weights determined by experiments and assigned to the cosine similarities, where $\alpha > 0$, $\beta > 0$, $\gamma > 0$, and $\alpha + \beta + \gamma = 1$;
- KWV is a keyword vector that consists of the keywords in the keyword set KY generated in Section 4.2;
- FSV is the first sentence vector, which is composed of the feature terms in the first sentence;
- OPV is the other sentence vector, which is composed of the feature terms in the other sentence.

Then, based on Eq. (10), the similarity of any two texts d_i and d_j is defined as the weighted sum of the cosine similarities of the corresponding vectors, that is,

$$\begin{aligned} sim(d_i, d_j) &= \alpha \times \cos(KWV_i, KWV_j) + \beta \times \cos(FSV_i, FSV_j) \\ &\quad + \gamma \times \cos(OPV_i, OPV_j) \\ &= \alpha \frac{\sum_{k=1}^{n} \vec{W}_{i,k} \times \vec{W}_{j,k}}{\sqrt{\sum_{k=1}^{n} \vec{W}_{i,k}^2} \sqrt{\sum_{k=1}^{n} \vec{W}_{j,k}^2}} \\ &\quad + \beta \frac{\sum_{r=1}^{m} \vec{W}_{i,r} \times \vec{W}_{j,r}}{\sqrt{\sum_{r=1}^{m} \vec{W}_{i,r}^2} \sqrt{\sum_{r=1}^{m} \vec{W}_{j,r}^2}} \\ &\quad + \gamma \frac{\sum_{s=1}^{l} \vec{W}_{i,s} \times \vec{W}_{j,s}}{\sqrt{\sum_{s=1}^{l} \vec{W}_{i,s}^2} \sqrt{\sum_{s=1}^{l} \vec{W}_{j,s}^2}}, \end{aligned} \tag{12}$$

where $\vec{W}_{i,k}$ and $\vec{W}_{j,k}$ represent the weights of the kth feature items of vectors KWV_i and KWV_j, respectively, and n is the total number of feature items in the two vectors.

Algorithm 3, FSCA, illustrates the functionality of the fine-grained similarity computation method.

Algorithm 3. Fine-grained Similarity Computation Algorithm (FSCA)

Require:
DD: The document set;
d_i: An text of the document set DD.
Ensure:
$sim_{FSC}(d_i, d_j)$: The similarity of the text d_i and d_j.
Remove d_i from DD
for each $d_j \in DD$ **do**
  Get $sim(d_i, d_j)$ by Eq. (12)
  $sim_{FSC}(d_i, d_j) \longleftarrow sim(d_i, d_j)$
end for
return $sim_{FSC}(d_i, d_j)$

4.4.2. Coarse-grained similarity computation

The centroid of every topic is used in the coarse-grained similarity computation. The centroid CD_i of top_i can be computed as:

$$CD_i = \frac{1}{N} \sum_{i=1}^{N} \vec{d}_i, \tag{13}$$

where N is the number of the text on one topic, and $\vec{d}_i$ is the feature vector for every text.

Based on the centroid CD_i and Eq. (10), the similarity between top_i and top_j is described as follows:

$$\begin{aligned} sim(top_i, top_j) &= \cos(CD_i, CD_j) \\ &= \frac{\vec{W}_{CD_i} \cdot \vec{W}_{CD_j}}{\left\|\vec{W}_{CD_i}\right\|_2 \times \left\|\vec{W}_{CD_j}\right\|_2}, \end{aligned} \tag{14}$$

where $\vec{W}_{CD_i}$ and $\vec{W}_{CD_j}$ are the weight vectors of centroids CD_i and CD_j.

Algorithm 4, CSCA, illustrates the functionality of the coarse-grained similarity computation method.

Algorithm 4. Coarse-grained Similarity Computation Algorithm (CSCA)

Require:
TS: The micro-topic set;
top_i: A topic of TS.
Ensure:
$sim_{CSC}(top_i, top_j)$: The similarity of the topic top_i and top_j.
Remove top_i from TS
Get CD_i by Eq. (13)
for each top_j in TS **do**
  Get CD_j by Eq. (13)
  Get $sim(top_i, top_j)$ by Eq. (14)
  $sim_{CSC}(top_i, top_j) \longleftarrow sim(top_i, top_j)$
end for
return $sim_{CSC}(top_i, top_j)$

The steps of the similarity computation in the single-pass clustering algorithm are shown in Algorithm 5. The thresholds given in our two similarity computation methods are different, and these thresholds are adjusted and determined based mainly on repetitive experimental experience.

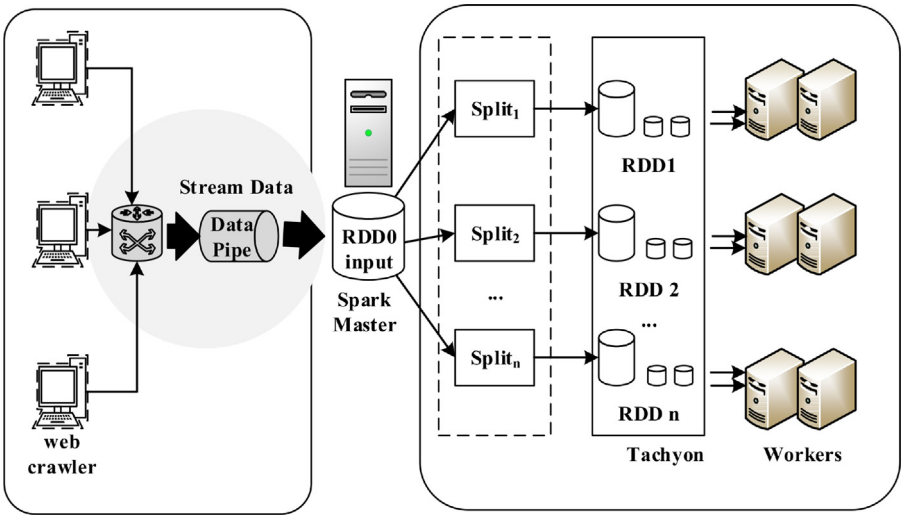

Fig. 4. The process of loading big data to the memory system.

Algorithm 5. Optimization Algorithm of Single-Pass (OASP)

Require:
DD: The document set;
TS: Topic seed set;
δ_1, δ_2: The thresholds in the similarity computation phase;
J: The similarity computation type.

Ensure:
TP: The final topic set.

if $J = 1$ **then**
  for each $d_i \in DD$ **do**
    $sim \longleftarrow \text{Max}(sim_{FSC}(d_i, d_j))$
    if $sim > \delta_1$ **then**
      $top_j \longleftarrow \{d_i\} \cup \{d_j\}$
    else
      $top_i \longleftarrow \{d_i\}$
      $TS \longleftarrow TS \cup \{top_i\}$
    end if
  end for
else
  for each $top_i \in TS$ **do**
    $sim \longleftarrow Max(sim_{CSC}(top_i, top_j))$
    if $sim > \delta_2$ **then**
      Remove top_i from TS
      Insert top_i into top_j
    end if
  end for
end if
$TP \longleftarrow TS$
return TP

5. Implementation on Spark

In order to cope with the challenges of big data, while improving the efficiency of the hot topic detection for microblog in big data, the Spark cloud computing framework is used to achieve the two-phase mic-mac hot topic detection.

5.1. Data partitioning

Before the hot topic detection process, the big data sets are loaded to the Spark Tachyon memory system as an *RDD* object. This process is presented in Fig. 4.

First, an *RDD* object RDD_0 is defined to save the original data set, which was obtained by a web crawler and stored in external files beforehand. Then, a random sampling technique is adopted to partition the RDD_0. Since a dataset can be evenly divided across partitions by a random partitioning method, we expect there to be an equal number of document objects in each partition. We divide the RDD_0 into n splits, where n is equal to the number of cores of the cluster. Let RDD_i, $1 \le i \le n$, denote the data sets partitioned from RDD_0, where n is the number of partitions, and $RDD_0 = \sum_1^n RDD_i$, $RDD_i \bigcap RDD_j = \emptyset$, $1 \le i \ne j \le n$. The n *RDD* objects are defined to save n subsets. *RDD* objects are constructed and saved into the Tachyon memory system.

5.2. Parallel process of TMHTD algorithm

To improve the speed of the hot topic detection process, a dual parallelization process is carried out. The whole computation flowchart of TMHTD on Spark is shown in Fig. 5, which consists of two tasks.

5.2.1. Task One: parallel text selection

First, as described above, the initial data is pre-processed by task one to obtain a data set, which can then be processed by the multi-worker evaluation scheme of the TMHTD algorithm. Fig. 6 shows the MapReduce process of text selection for document subsets. From Fig. 6, it can be seen that the process of the algorithm is implemented in the Spark computing cluster with the MapReduce model with RDD objects. K subsets are allocated to K map tasks at the same time, and K map tasks are assigned to multiple slave nodes. At the map stage, as we can see from Fig. 6, each feature variable subspace is mapped to the corresponding map task concurrently. The $CF(f_{i,k})$ and $w_{i,k}$ information values of each feature variable subspace are calculated in map tasks at the same time.

Second, it goes to the shuffle stage. The $Score(f_{i,k})$ of all feature variables is shuffled and calculated.

Third, it proceeds to reduce stage, in which the weighted value $Score(f_{i,k})$ of each feature variable is combined with the same variable name. Then, each value $Score(d_i)$ is calculated and sorted by a corresponding reduction function.

Finally, the score values are sorted in descending order, and the top m values are selected as the final results of the document data. The number of dimensions of the document sets is reduced from M to m with the first task. The dimension subspace of document data sets is saved as a new *RDD* object *RDD′*. The steps of the parallel text selection process are presented in Algorithm 6.

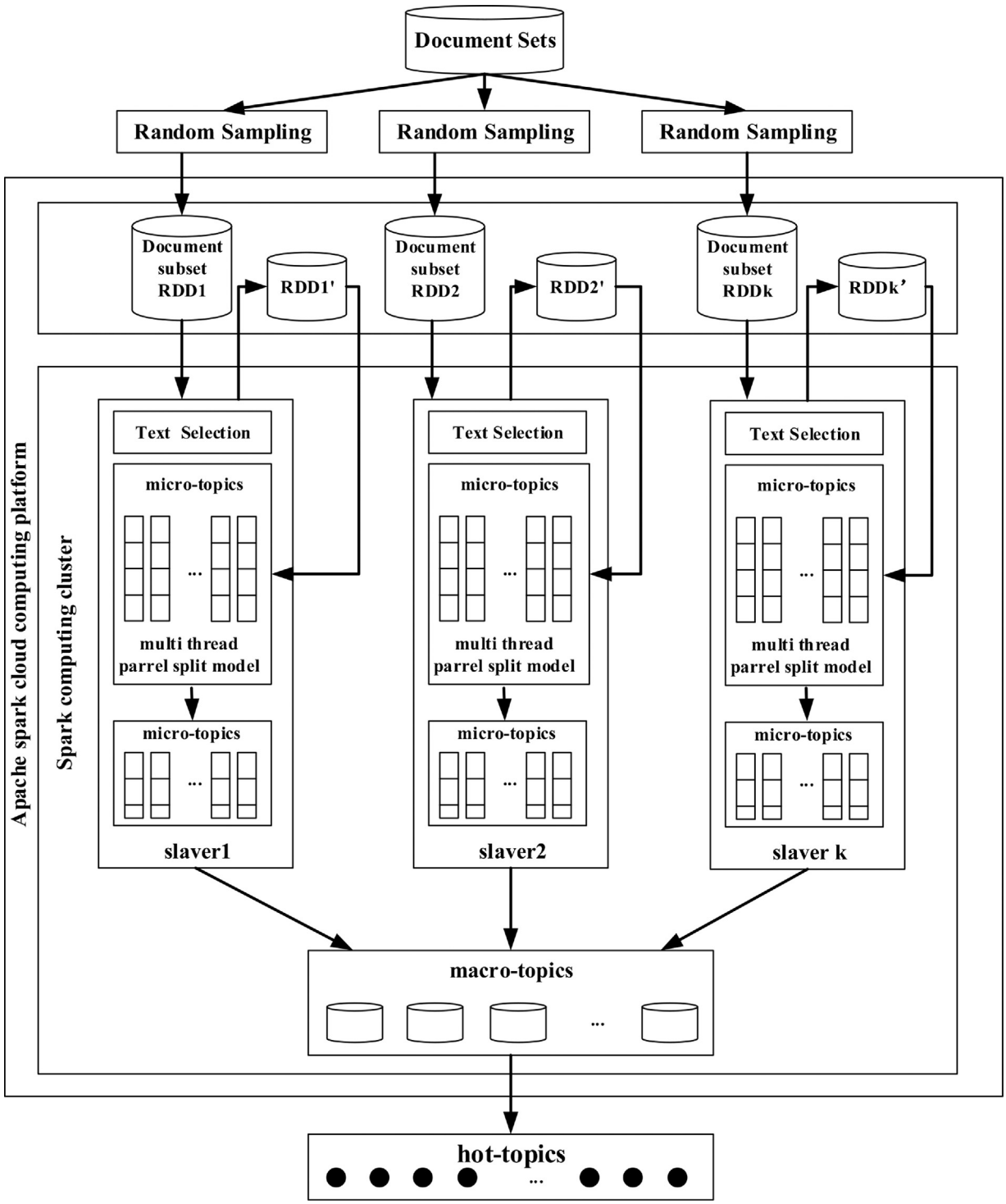

Fig. 5. The parallel process of TMHTD approach.

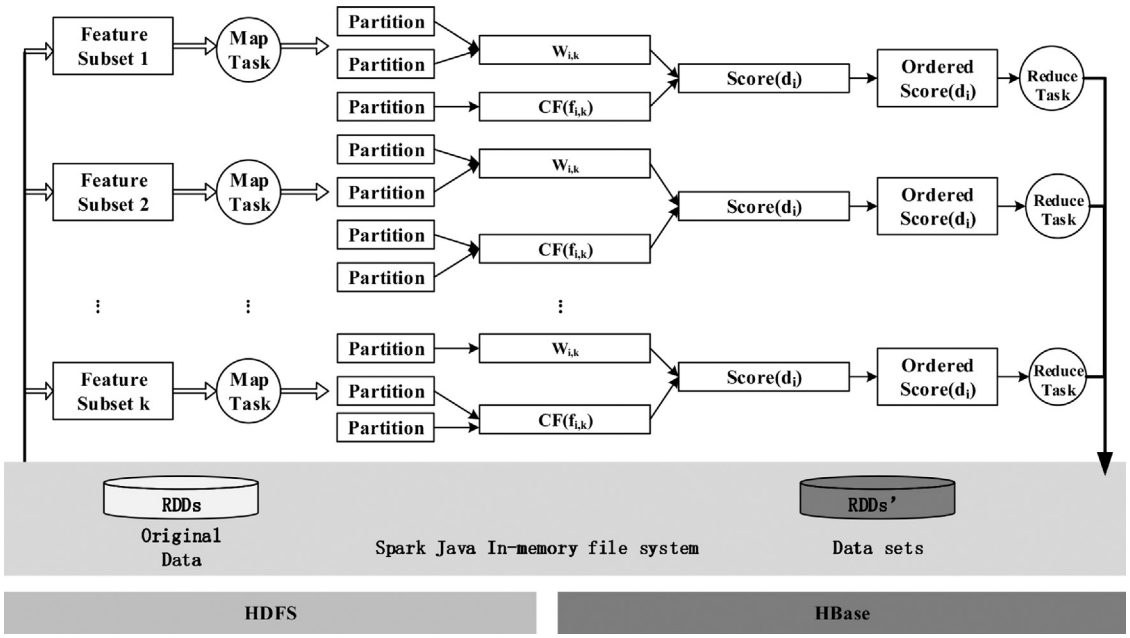

Fig. 6. The MapReduce process of parallel text selection.

Algorithm 6. Parallel Process of Text Selection (PPTS)

Require:
RDD_0: *RDD* objects of document set *D*.
Ensure:
RDD'_j: The *RDD* objects after text selection.
//Transformation
for *RDD* j=1 to k **do**
  for each feature variable $f_{i,k} \in RDD_j$ **do**
    $l \longleftarrow i \times k$
    Map l feature subspace of RDD_j to $<f_{i,k}, RDD_{j,i}>$
    Calculate the weight value $< f_{i,k}, w_{i,k} >$ with Partition.w($RDD_{j,i}$)
    Calculate the contribution value $<f_{i,k}, CF(f_{i,k})>$ with Partition.cf($RDD_{j,i}$)
    Calculate the score value $<f_{i,k}, Score(f_{i,k})>$ with function Scorevalue($<f_{i,k}, RDD_{j,i}>$)
  end for
end for
//Action
for *RDD* j=1 to k **do**
  Shuffle ($<f_{i,k}, Score(f_{i,k})>$)
  Calculate the score value score(d_i) by function Reduce.getscore(d_i, $Score(f_{i,k})$)
  Assign the m texts with top m score value to $RD[0, \ldots, m-1]$
  $RDD'_j \longleftarrow RD[0, \ldots, m-1]$
end for
return RDD'_j

5.2.2. Task two: parallel clustering

The second MapReduce task receives the output of reduce from task one as an input. It aims to detect the final hot topics. The MapReduce process of the parallel clustering algorithm is presented in Fig. 7.

As described above, the RDD' subsets should be assigned to a plurality of slave nodes on the Spark platform. At the map stage, k hot topics are built in each map task at the same time with the clustering algorithm (Algorithm 5) and topic selection algorithm (Algorithm 2). The selected hot topics TS_1 and removed hot topics TS_2 are calculated at the partition stage. At the reduce stage, TS_1 are shuffled into one reduce job. Next, the final macro-topics are calculated using Algorithm 5. Based on the output of these steps, we can present a list of hot topics. The steps of the parallel clustering are presented in Algorithm 7.

Algorithm 7. Parallel Process of Clustering Algorithm (PPCA)

Require:
RDD'_i: A RDD_j object after text selection.
Ensure:
TOP: The hot topic set.
for *RDD'* i=1 to k **do**
  Map RDD'_i to $< i, RDD'_i >$
  Get the micro-topic *TS* from $< i, RDD'_i >$ using Algorithm 5 (OASP(RDD'_i, TS_0, δ_1, δ_2, 1))
  Get the micro-topic set *TSS* using Algorithm 2 (Macro-TSA(*TS*, δ))
  $TS_1 \longleftarrow TSS$
end for
Combine the topic *TP* with reduce.*TSS*(TS_1)
Get the macro-topic *TP'* from *TP* using Algorithm 5 (OASP(RDD'_i, *TP*, δ_1, δ_2, 2))
Calculate the topic heat value $Heat(top_i)$ by Eq. (8) from *TP'*
Sort M topic variables of *TP'* in descending order by $Heat(top_i)$
Put top m hot topics to $TOP[0, \ldots, m-1]$
return *TOP*

6. Experimental evaluation

6.1. Experiment setup and datasets

Our data were gathered from the Sina Weibo which is the top microblog service in China. We monitored a stream using the Sina Weibo API to crawl and collect approximately 16 million pieces microblog data from 800 thousand users. This access level collects vast amounts of data spread over a short time horizon and provides a statistically significant input. Each microblog was saved as a text file, and the maximum size of the dataset is 3 TB. The size of the dataset is increased to different levels in each set of experiments.

We have conducted our experiments on LuCloud, which is a homogeneous Spark cluster located in Hunan University running the stable version Spark 1.1.0. The cluster consists of one master computing node and 100 slave computing nodes. Each computing node has the same physical environment, i.e., one Pentium (R) Dual-Core 3.20 GHZ CPU and 8 GB memory. The operating system of each node is Ubuntu 12.04.4. All the nodes are interconnected with a high speed Gigabit network.

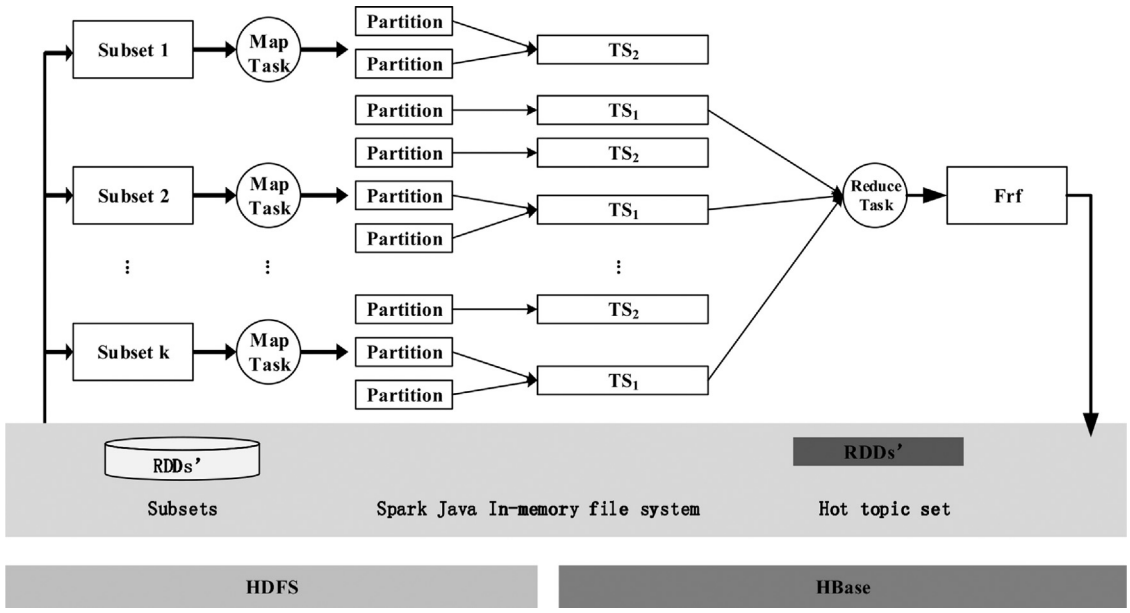

Fig. 7. The MapReduce process of parallel clustering algorithm.

6.2. Accuracy evaluation

We study the accuracy of our algorithm. To quantify accuracy, the experimental results are analyzed with the *F*-measure (F_m):

$$F_m = \frac{(1+\beta^2)(recall \times precision)}{\beta^2 \times (recall + precision)}, \tag{15}$$

where *F* denotes the synthetic performance of a system, *precision* is the percentage of correct detections, and *recall* is the percentage of the total hot topics detected that are successfully detected. The value of β is taken as 1. The higher the *F*-measure, the more accurate the final result is. The value of β is the relative weight between *recall* and *precision*, the value of β is more than 1 if *precision* is more important and the value of β is less than 1 if *recall* is more important. We set the value to 1, and consider *recall* and *precision* to be equally important in our experiments. *Recall* is defined as Eq. (16):

$$recall = \frac{N_{right}}{N_{all}}, \tag{16}$$

where N_{right} denotes the number of texts that are correctly classified into a topic, and N_{all} denotes the number of all texts in a topic. *Precision* is defined as Eq. (17):

$$precision = \frac{N_{right}}{N_{reality}}, \tag{17}$$

where $N_{reality}$ denotes the number of texts which are actually classified into a topic. To evaluate the experiment by means of *precision*, *recall* and *F-measure*, the existing algorithm is used to detect the hot topics of the available data and obtain the hot topics and preliminary classification of the data. Then, we manually label the classified data.

We evaluate the accuracy of the TMHTD algorithm by comparing it to the general algorithm and other improved algorithms. Table 1 lists the names and related descriptions of all algorithms used in the experiments. HTD-LDA is an improved approach in Ref. [10], which is one of the successful and popular topic modeling algorithms in practice. Some experiments are carried out using the HTD-SP, HTD-LDA, and TMHTD algorithms. The experiment using HTD-SP was carried out on a single machine. In the experiments using HTD-LDA and TMHTD, Spark 1.1.0 was deployed in a cluster with 5 nodes.

Table 1
All algorithms used in experiments.

Algorithms	Description
HTD-SP	The general approach (single-pass)
HTD-LDA	The improved approach in Ref. [10]
TMHTD	The efficient approach in this paper

Table 2
Results of three algorithms.

Data size	Methods	Precision	Recall	F_m
32 MB	HTD-SP	0.7572	0.8715	0.8103
	HTD-LDA	0.8194	0.9159	0.8650
	TMHTD	0.9036	0.9424	0.9226
64 MB	HTD-SP	0.7036	0.8228	0.7585
	HTD-LDA	0.7687	0.8574	0.8106
	TMHTD	0.8774	0.9067	0.8918
1 GB	HTD-SP	0.6437	0.7292	0.6838
	HTD-LDA	0.6966	0.7546	0.7244
	TMHTD	0.8363	0.8430	0.8396
4 GB	HTD-SP	0.5195	0.5654	0.5418
	HTD-LDA	0.5422	0.6571	0.5941
	TMHTD	0.7372	0.8016	0.7680

Each case uses a different dataset size. The sizes of the datasets are 32 MB, 64 MB, 1 GB, and 4 GB. Through calculating the *F*-measures of the algorithms, the different accuracies of various environments are compared and analyzed.

The results are shown in Table 2. It is found that the $F_{measure}$ values of HTD-SP and HTD-LDA are much lower than those of TMHTD. The F_m values of HTD-SP and HTD-LDA are, respectively, 0.7585 and 0.8106, lower than that of TMHTD (0.8918) when the data size is 64 MB. The F_m values of HTLD-SP and HTD-LDA are, respectively, 0.5418 and 0.5941 lower than that of TMHTD (0.7680) when the data size is 4GB. It is found that because there are noisy data in the experimental data sets, when the data size increases, the average F_m of each algorithm is reduced gradually. Generally, TMHTD can provide more accurate detection than traditional methods (for example, HTD-SP and HTD-LDA). Thus, hot topics detected by our method will better satisfy users.

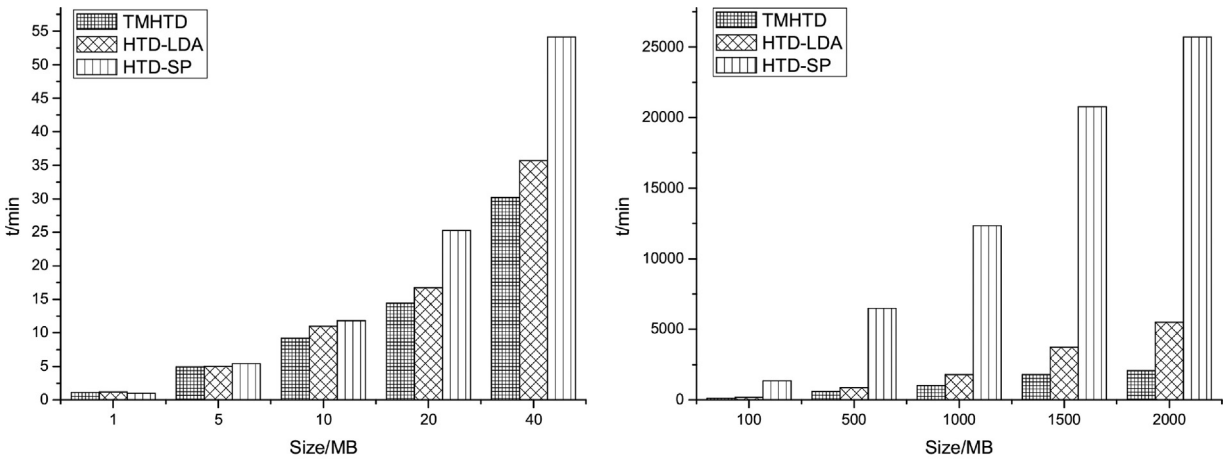

Fig. 8. Varying the data set size for run time.

6.3. Performance evaluation

Two well-accepted performance metrics, run time and speedup, are adopted to measure the performance of TMHTD. Speedup refers to how much faster a parallel algorithm is than the corresponding sequential algorithm, which can be defined as follows:

$$\varphi = \frac{T_1}{T_p}, \tag{18}$$

where T_1 is the sequential execution time and T_p is the parallel execution time. If the speedup has a linear relation with the number of nodes when the data size is fixed, the algorithm will have good scalability.

6.3.1. Varying data set size for run time

Our experiments are performed with varied data sizes on a Spark cluster with 15 nodes and on a single machine. Here the TMHTD and HTD-LDA methods are performed on a Spark cluster, and the HTD-SP method is performed on a single machine. In order to clearly observe the results, our experimental results are divided into several charts to show different results. In Fig. 8, there are three charts presenting the run time comparisons of the TMHTD, HTD-LDA, and HTD-SP.

We use 1 MB, 5 MB, 10 MB, 20 MB, and 40 MB random sampling to change the data set size for TMHTD, HTD-LDA, and HTD-SP. Fig. 8(a) illustrates the three algorithms in a small instance. It is clear that increasing the size of the random sampling leads to increased run time, and the degree of growth between TMHTD and HTD-SP is quite different. However, there is not much difference between the run times of the three algorithms.

Fig. 8(b) illustrates the effects on three algorithms of varying the data set size, using 100 MB, 500 MB, 1000 MB, 1500 MB, and 2000 MB random sampling. The average running time of TMHTD is less than those of HTD-SP and HTD-LDA. Therefore, TMHTD achieves a faster processing speed than the other algorithms in our experiments. When the size of the sample data is increased, the tendency is more noticeable. When the sample data size grows from 100 MB to 2000 MB, the runtime of HTD-SP is increased from 1356.16 to 25709.05 s, while that of TMHTD is increased from 129.43 to 2075 s. We also observe that the run time of the HTD-SP method keeps increasing, and the degree of growth is increasing. While using the parallel method, the run time has modest growth especially for large-scale samples.

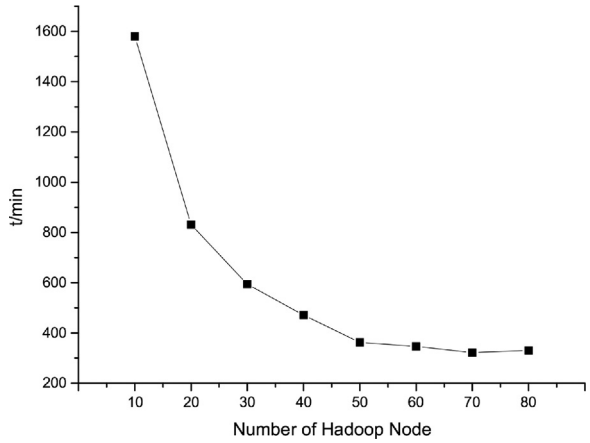

Fig. 9. Varying the Spark nodes with the same data set size.

From the two groups of experimental results, we know that the operating speed of the TMHTD algorithm is higher than those of HTD-SP and HTD-LDA.

6.3.2. Varying the number of computing nodes for run time

For the same data size (1000 MB random samples), we vary the number of Spark nodes in the TMHTD algorithm. Through observing the run time of the algorithm, the different performances for various environments are compared and analyzed. Fig. 9 shows the changes in the run time when the number of cluster nodes is increased gradually.

In Fig. 9, we find that increasing the number of cluster nodes from 10 to 50 significantly reduces the run time. This demonstrates that the number of Spark nodes affect the time efficiency of TMHTD. As we know, the computing time, the disk I/O time and the network I/O time are the major parts of the total execution time of big data processing jobs. For Spark, the distributed memory data abstraction reduces the impact of the disk I/O time on performance. However, when the number of Spark nodes is increased from 50 to 80, the total running time shows no obvious decrease. We can observe that for more than 50 nodes, the average run time of the TMHTD algorithm has a decreasing tendency, but computing nodes have little impact on computing performance. This happens because too many map tasks lead to over-splitting of the data, introducing too much overhead and internal communication delays. When the number of cluster nodes continues to increase, the disk I/O operations of each

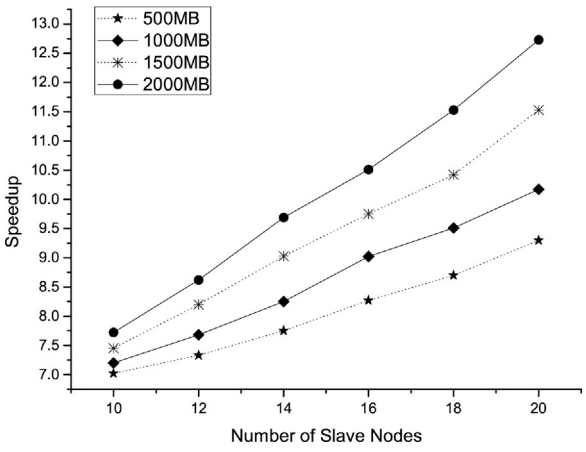

Fig. 10. Varying the Spark nodes with the same data set size.

cluster node cannot be raised for reading or writing the intermediate data.

6.3.3. Speedup of the TMHTD algorithm with different data sets in different environments

To verify the speedup of TMHTD, experiments are conducted in size a cluster of nodes ranging from 10 to 20. There are four synthetic data sets used in the experiments (500 MB, 1000 MB, 1500 MB and 2000 MB data size). Fig. 9 shows the speedup of TMHTD.

From Fig. 8, we can observe that when facing a large amount of data, our algorithm can achieve larger speedup. This is primarily because the computation cost is so dominant that the effect of the communication cost on speedup is almost invisible. The speedup value is 7.02 when the data size is 500 GB and the number of nodes is 10, while the speedup value reaches 7.73 when the data size is 2000 GB and the number of nodes does not change. Meanwhile, the speedup of TMHTD increases relatively linearly with the growth of the number of nodes. The experimental result shows that TMHTD on the Spark platform has good speedup with big data and performs better with larger data sets (Fig. 10).

Overall, these experimental results show that TMHTD performs well in terms of accuracy, and TMHTD on the Spark platform has good performance in a big data environment.

7. Conclusion

In this paper, we have investigated the problem of how to quickly and accurately obtain valuable hot topics from a large number of textual digital materials, and proposed a non-iterative parallel algorithm called two-phase mic-mac hot topic detection (TMHTD) method, which is specially designed for microblog under big data environment, and implemented on the Apache Spark. The proposed TMHTD method consists of two phases: the micro-clustering phase and the macro-clustering phase. In the first phase, original data sets are partitioned into a group of smaller data subsets by using TMHTD algorithm, and then these data subsets are clustered into many topics by using an improved non-iterative single-pass cluster algorithm. In the second phase, these topics are integrated into a single data set, and then the data set is clustered again according to an improved non-iterative single-pass clustering algorithm to obtain the final detection results. In order to handle large-scale data, a set of MapReduce operations are designed to achieve hot topic detection. In order to effectively improve the detection accuracy of hot topic for high-dimensional, large, and noisy data, with the parallel processing, three optimization methods are also proposed. A large number of experimental results show that the performance and accuracy of the TMHTD algorithm are significant improved over previous methods. In future work, we will further research how to consider users' different preferences and meet users' personalized requirements, exploiting other Ensemble Classification and Regression methods [43,44], which are used in hot topics detection in the big data environment in Spark.

Acknowledgments

The authors acknowledge four anonymous reviewers for their constructive comments. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), and the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124).

References

[1] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with big data, IEEE Trans. Knowl. Data Eng. 26 (1) (2014) 97–107.
[2] K.-Y. Chen, L. Luesukprasert, S.-C.T. Chou, Hot topic extraction based on timeline analysis and multidimensional sentence modeling, IEEE Trans. Knowl. Data Eng. 19 (8) (2007) 1016–1025.
[3] L. Zhang, P.N. Suganthan, A survey of randomized algorithms for training neural networks, Inf. Sci. 364–365 (2016) 146–155.
[4] S. Vieweg, A.L. Hughes, K. Starbird, L. Palen, Microblogging during two natural hazards events: what twitter may contribute to situational awareness, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2010, pp. 1079–1088.
[5] D. Zhao, M.B. Rosson, How and why people twitter: the role that micro-blogging plays in informal communication at work, in: Proceedings of the ACM 2009 International Conference on Supporting Group Work, ACM, 2009, pp. 243–252.
[6] G. Mishne, J. Dalton, Z. Li, A. Sharma, J. Lin, Fast data in the era of big data: Twitter's real-time related query suggestion architecture, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 1147–1158.
[7] X. Yun, G. Wu, G. Zhang, K. Li, S. Wang, Fastraq: a fast approach to range-aggregate queries in big data environments, IEEE Trans. Cloud Comput. 3 (2) (2015) 206–218.
[8] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, R. Harshman, Indexing by latent semantic analysis, J. Am. Soc. Inf. Sci. 41 (6) (1990) 391.
[9] T. Hofmann, Probabilistic latent semantic indexing, in: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 1999, pp. 50–57.
[10] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (Jan) (2003) 993–1022.
[11] Q. He, K. Chang, E.-P. Lim, A. Banerjee, Keep it simple with time: a reexamination of probabilistic topic detection models, IEEE Trans. Pattern Anal. Mach. Intell. 32 (10) (2010) 1795–1808.
[12] L. AlSumait, D. Barbará, C. Domeniconi, On-line lda: adaptive topic models for mining text streams with applications to topic detection and tracking, in: 2008 Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 3–12.
[13] D. Newman, E.V. Bonilla, W. Buntine, Improving topic coherence with regularized topic models, Advances in Neural Information Processing Systems (2011) 496–504.
[14] Q. Diao, J. Jiang, F. Zhu, E.-P. Lim, Finding bursty topics from microblogs, in: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers, vol. 1, Association for Computational Linguistics, 2012, pp. 536–544.
[15] M.S. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, E.H. Chi, Eddi: interactive topic-based browsing of social status streams, in: Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology, ACM, 2010, pp. 303–312.
[16] W.X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, X. Li, Comparing twitter and traditional media using topic models, in: European Conference on Information Retrieval, Springer, 2011, pp. 338–349.
[17] B. Huang, Y. Yang, A. Mahmood, H. Wang, Microblog topic detection based on LDA model and single-pass clustering, in: International Conference on Rough Sets and Current Trends in Computing, Springer, 2012, pp. 166–171.
[18] Y. Chen, H. Amiri, Z. Li, T.-S. Chua, Emerging topic detection for organizations from microblogs, in: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2013, pp. 43–52.
[19] G. Ge, L. Chen, J. Du, The research on topic detection of microblog based on TC-LDA, in: 2013 15th IEEE International Conference on Communication Technology (ICCT), IEEE, 2013, pp. 722–727.
[20] X. Cheng, X. Yan, Y. Lan, J. Guo, BTM: topic modeling over short texts, IEEE Trans. Knowl. Data Eng. 26 (12) (2014) 2928–2941.
[21] H. Tu, J. Ding, An efficient clustering algorithm for microblogging hot topic detection, in: 2012 International Conference on Computer Science & Service System (CSSS), IEEE, 2012, pp. 738–741.

[22] W. Ai, D. Li, Parallelizing hot topic detection of microblog on spark, International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (2016) 1461–1468.
[23] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, HotCloud (2010) 10.
[24] X. Lin, P. Wang, B. Wu, Log analysis in cloud computing environment with Hadoop and Spark, in: 2013 5th IEEE International Conference on Broadband Network & Multimedia Technology (IC-BNMT), IEEE, 2013, pp. 273–276.
[25] G. Zhou, D. Zhao, K. Zou, W. Xu, X. Lv, Q. Wang, W. Yin, The static security analysis in power system based on spark cloud computing platform, in: Smart Grid Technologies-Asia (ISGT ASIA), 2015 IEEE Innovative, IEEE, 2015, pp. 1–6.
[26] M. Solaimani, M. Iftekhar, L. Khan, B. Thuraisingham, Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source VMware performance data, in: 2014 IEEE International Conference on Big Data (Big Data), IEEE, 2014, pp. 1086–1094.
[27] G. Privitera, G. Ghidini, S.P. Emmons, D. Levine, P. Bellavista, J.O. Smith, Soft real-time GPRS traffic analytics for commercial m2m communications using spark, in: 2014 International Conference on Smart Computing (SMARTCOMP), IEEE, 2014, pp. 13–20.
[28] A.K. Koliopoulos, P. Yiapanis, F. Tekiner, G. Nenadic, J. Keane, A Parallel Distributed Weka Framework for Big Data Mining Using Spark, 2015, pp. 9–16.
[29] F. Zhang, M. Liu, F. Gui, W. Shen, A. Shami, Y. Ma, A distributed frequent itemset mining algorithm using spark for big data analytics, Cluster Comput. 18 (4) (2015) 1493–1501.
[30] Y. Liang, J. Liu, C. Fang, N. Ansari, Spark-based large-scale matrix inversion for big data processing, Computer Communications Workshops (2016) 718–723.
[31] W. Liu, Q. Li, Y. Cai, Y. Li, X. Li, A prototype of healthcare big data processing system based on spark, International Conference on Biomedical Engineering and Informatics (2016) 516–520.
[32] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, Apache spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.
[33] M.A. Alsheikh, D. Niyato, S. Lin, H.P. Tan, Z. Han, Mobile big data analytics using deep learning and apache spark, IEEE Netw. 30 (3) (2016) 22–29.
[34] O.C. Marcu, A. Costan, G. Antoniu, M.S. Perezhernandez, Spark versus flink: understanding performance in big data analytics frameworks, IEEE International Conference on CLUSTER Computing (2016) 433–442.
[35] A.G. Shoro, T.R. Soomro, Big Data Analysis: Apache Spark Perspective, vol. 15, 2015.
[36] K.K. Bun, M. Ishizuka, Topic Extraction from News Archive Using TF*PDF Algorithm, 2002, pp. 73–82.
[37] G. Salton, Automatic Text Processing: The Transformation, Analysis, and Retrieval of, Addison-Wesley, Reading, 1989.
[38] Y. Zhu, Y. Hu, Enhancing search performance on Gnutella-like P2P systems, IEEE Trans. Parallel Distrib. Syst. 17 (12) (2006) 1482–1495.
[39] R. Papka, J. Allan, On-line New Event Detection Using Single Pass Clustering Title2, 1998.
[40] Aparche, Spark, 2015, January http://spark-project.org.
[41] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Trans. Parallel Distrib. Syst. PP (99) (2016) 1.
[42] C. Yang, J. Yang, H. Ding, H. Xue, A Hot Topic Detection Approach on Chinese Microblogging, 2013.
[43] Y. Ren, L. Zhang, P.N. Suganthan, Ensemble classification and regression—recent developments, applications and future directions, IEEE Comput. Intell. Mag. 11 (1) (2016) 41–53 (review article).
[44] A. Rajasekhar, N. Lynn, S. Das, P.N. Suganthan, Computing with the collective intelligence of honey bees – a survey, Swarm Evol. Comput. (2016).