2F3-04

A Scalable Real-Time Burst Detection System based on Kleinberg Algorithm

[†]Min Luo, [†]Akihiro Yamanaka, [†]Masato Sawada, [†]Satoshi Oda, [‡]Akihiro Shiozawa, [‡]Naoto Hirano, [†]Keitaro Horikawa

[†]Software Innovation Center
NTT R&D

[‡] Mathematical Systems Inc.
NTT DATA

Real-time burst detection is an important topic in data stream analysis. An effective way to find burst events is using a sliding time-window to record the frequency of evens that happened in each slide interval. Although many burst detection methods have been conducted based on this idea, only a handful of them are suitable for real-time detection. In this paper, we propose a real-time burst detection system based on Kleinberg algorithm and Jubatus. It is able to detect 100 events' burst whose arriving frequency is of several milliseconds. By deploying our proposal on Jubatus framework, 10x times larger numbers of events could be detected at the same accuracy with the same regular specs of PCs. According to our knowledge, this is the first system for scalable real-time burst detection. Experimental evaluations of our prototype system on tweets and financial data analysis demonstrate the feasibility and scalability of our proposal.

1. Introduction

In time-series data processing, one important indication of change is the presence of 'burstiness'. There has been a great deal of research proposed for identifying and tracking burst. And recently, there is a growing requirement in real-time burst detection in lots of application scenarios, like monitoring facility sensor data, traffic control systems, and stock trading systems.

A burst is defined as an abnormal aggregate in the stream in a short period of time. For example, the popular keywords extracted from document streams may be a valid indicator of emerging or changing topics of general interest; a volume surge in stock trading usually implies a strong buying/selling signal [1]. Burst detection problem is concerned with identification of such bursts, providing useful insights into the unusual events and in turn facilitating timely decision making.

Intuitively, bursted period in a data stream can be defined as time intervals in which the data value exceeds a predetermined threshold value. Previous studies have presented different ways of determining the threshold [1, 2]. In practice, however, the use of predefined thresholds might make it difficult to detect burst activity. For example, the occurrences of a given feature may oscillate above or below the threshold in certain noisy data streams. Some of the bursts may be easily recognized as one long burst period by humans; however, threshold-based algorithms may identify them as several short period bursts or no burst at all.

To alleviate this problem, Kleinberg [3] has developed a framework for doing this. It defines a word burst as an increase in the occurrence (arrival rate) of the word in a stream of text, and developing an automaton for tracking an optimized estimate of this rate, by an alternative state-based method using the Hidden Markov Model (HMM). This type of method is a natural extension of the threshold-based methods with a more relaxed and variable notation of threshold. Kleinberg's burst model has inspired many subsequent efforts, since his traffic-based burst definition is intuitively appealing.

However, because original Kleinberg is a batch based burst detection algorithm, the burst of a word is determined by analyzing its appearance frequency and calculating its burst state transfer cost in the whole stream text. This full-length calculation could be a pretty high overhead for real-time burst detection. Therefore, all the previous researches based on original Kleinberg will inherent its shortages in real-time applications.

Our research is motivated to improve original Kleinberg detection solution for real-time burst detection. In addition, our approach considers the distributed parallel processing capability, and we also implemented our proposal on Jubatus framework [4], so as to support high scalability in multiple event detection.

2. Previous Work

2.1 Kleinberg's Burst Model

Motivated originally by a problem of representing bursts of email messages, Kleinberg's burst algorithm [3] models bursts with an infinite state automaton, in which each state represents a message arrival rate. The higher the state, the smaller the expected time gap between messages. `Word bursts' can then be defined as having arrival rates defined by the number of messages containing a particular word. Additionally, jumping from a lower state to a higher state has an associated cost, while the cost to drop down from a higher state to a lower state is 0. Formally, these states are determined by:

For a time series $z = \{z_t \mid t = 0, 1, \ldots, n\}$ of inter-arrival gaps, find a state sequence $q = \{q_{i_t} \mid t = 0, 1, \ldots, n\}$ minimizing cost function:

$$c(q|z) = b(q)\ \ln\left((1-p)/p\right) + \left(\sum_{t=0}^{n} -\ln f_{i_t}(z_t)\right)$$

Where p is the probability of a state change, b($\boldsymbol{q}$) is the number of state transitions (changes in successive states) in **q**, and $f_i(z) = a_i * e^{-aiz}$ is the exponential density function for gap values z with arrival rate a_i.

Contact: Min Luo, Software Innovation Center, NTT R&D, 3-9-11 Midoricho, Musashino-shi Tokyo, 180-8585, Japan, luo.min@lab.ntt.co.jp, Tel: 0422-59-2746

By finding the optimal sequence of states minimizing the cost of transitions and the cost of differences between real arrival rate and the predicted emission rate, a time series of burst strengths is obtained. However, the complexity of the algorithm for finding this optimal sequence itself can be pretty large.

2.2 Features Study of Kleinberg's Burst Model

In this sub-section, we focused on studying the features of Kleinberg model. We first compare it with the 'event number threshold based' burst detection solutions, such as [1, 2]; and the 'Change Finder' solution, which is based on a widely used AR (Auto Regressive) model in real-time data analysis [5].

For an easy understanding of the pros and cons in Kleinberg model, we conclude our comparison result in Table I and Table II.

In Table I, the 'time resolution' means the time range within which period an event burst is detected; Kleinberg has the best result because all the others have to manually determine an appropriate time resolution based on the event's nature. The 'threshold setting' means the necessity in setting threshold value for a burst decision, the 'easy of interpretation' means whether the burst results are easy to understand, and the 'sensitivity for small burst' means the ability in detecting a small degree of burst. Because only the 'event number based' solutions require a predetermined threshold value, which often results in low detection accuracy especially for detecting small burst, it gets the worst results in these three features. The 'accuracy of detected time' means the latency in burst detection, and the 'calculation overhead' means the calculation cost during burst detection. Because 'Change Finder' has two sequential smoothing phases during burst calculation, it results the latency in detecting burst and a relatively high calculation cost. On the other hand, calculation cost in Kleinberg is even higher O(log(n) ^n), which cause the comparison results in these two features. The 'strength of noise' means the immunity from detecting noise bursts. 'Change Finder' gets the best result because of its two phases smoothing processing. Kleinberg may also achieve different noise immunity by change its state transfer cost definition.

	event number based	Change Finder	Klein berg
time resolution	△	△	○
threshold setting	×	○	○
easy of interpretation	△	○	○
sensitivity for small burst	×	△	○
accuracy of detected time	○	△	○
calculation overhead	○	△	×
strength of noise	×	○	△

Table I. Feature Comparisons with Rivals

Kleinberg model [5] itself contains two methods for burst detection, the 'Continuously Bursts' (CB) and the 'Enumerating Bursts' (EB). We conclude their different features in Table II.

In CB detection method, burst is detected once an event is happened. Its burst degree grows as its happening density increased. The density is defined as the happening time interval of a specified event after its previous occurring time. While in EB detection method, timeline is divided in to *m* windows, and burst detection is executed for the events happened in the latest time window that has just past. The burst degrees depend on the percentage of target events among all the events those happened in every past *m* windows.

Because burst detection is executed for every occurrence of an event right after its happening, EB has the better burst time resolution. As a consequence, its calculation cost is O(log(n) ^n), which is much higher than that in CB O(2^m). In addition, for a specified observation time interval, the memory used by CB is un-predictable, while EB has a fixed memory usage that is proportional to the number of time windows. We conclude the CB and EB comparison in Table II.

	CB	EB
time resolution	○	×
calculation overhead	×	○
memory usage management	×	○
multiple events detection	×	○

Table II. Features of two Kleinberg methods

3. Proposed Method

As we have analyzed in Section 2, Kleinberg overwhelms other burst detection solutions in almost all the key performance features, except for its calculation overhead. Thus, it is important to optimize the calculation overhead so as to inherit its other features for large scale real-time burst detection applications.

Note that, EB method has the lower burst degree calculation complexity, and fixed burst window size as well as its memory consumption during calculation. In this section, we focus on optimizing EB based Kleinberg model in this work, because its capability in utilizing a distribution processing framework for parallel processing and high scalability.

3.1 Re-definition of Burst Level

In original Kleinberg EB detection method, only two values are used to demonstrate a burst's level (0: not bursted; 1: bursted), which is impossible for users to understand the intensity of a burst. On the other hand, there defines a burst's *weight* for a continuous period of bursted batches. It is defined as the integration weight value of all the bursted batches, and is used in calculating the burst level of batches, which is:

$$\Sigma (\sigma(0, r\_t, d\_t) - \sigma(1, r\_t, d\_t))$$

To intuitively interpret detected burst, we re-define the burst levels from original (0, 1) with the *weight* value

$$\text{Maximum } \{(\sigma(0, r\_t, d\_t) - \sigma(1, r\_t, d\_t)), 0\}$$

This re-definition does not introduce new calculation cost into Kleinberg, and we will demonstrate that this new burst level may accurately illustrate burst status and improve the result interpretation ability of Kleinberg later in our experiments.

3.2 Optimizing Burst Calculation Cost

In original Kleinberg definition, to calculate an event's burst level in a batch interval, it requires the event's burst levels and occurrence percentages in all previous batches. This may become a huge calculation overhead O(2^m) when handling endless data streams, where the number of previous batches *m* keeps increasing. To restrain the calculation cost for endless data stream, we introduce the concept of *batch windows*.

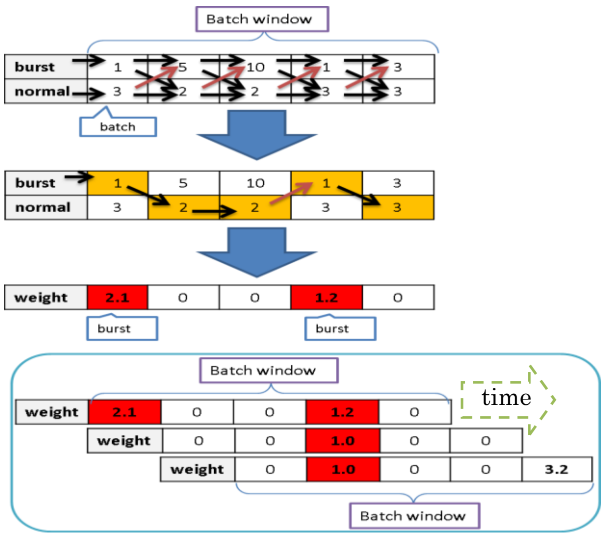
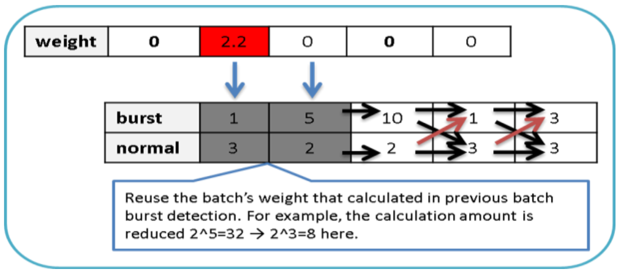


Figure 1. An Image of Batch Windows

As shown in Figure 1, by setting a batch window for burst detection, burst weight calculation after a new batch interval is executed by calculating the burst status and status transfer overhead between the latest *m* adjacent batches in the window. For example, when batch window size is *m* = 5 as shown in Figure 1, the calculation overhead is limited to 2^5 steps. When time moves to the next batch interval, the burst window slides to discard the oldest batch and employ the latest batch for the new burst weight calculation when current batch interval finishes.

Based on this batch window, we further propose a calculation cost reduction method. In this method, we skip the weight calculation for the oldest m' batches by reusing their weight results calculated in previous detection. This method is able to reduce the detection latency for short interval burst detection application, and guarantees a relatively long batch window for higher noise strength. For the length limitation, please also refer the details in Figure 2, where m'=2.



Reuse the batch's weight that calculated in previous batch burst detection. For example, the calculation amount is reduced 2^5=32 → 2^3=8 here.

Figure 2. Cost Reduction Proposal

We will demonstrate the effectiveness of this cost reduction proposal later in our experiments.

4. Implementation on Jubatus

Although our revised Kleinberg method greatly reduces the burst detection cost for a single event, a single server's resource may easily be saturated when dealing with multiple events' burst detection in parallel. Therefore, it is important to provide the revised method in a scalable framework, where multiple events' detection workload are shared and processed on several servers.

Jubatus [4] is a distributed processing framework and streaming machine learning library. It contains a scalable framework for real-time big data analysis. By using the sliding windows module and MIX mechanism that already implemented in Jubatus, we could easily deploy the proposed method onto Jubatus framework.

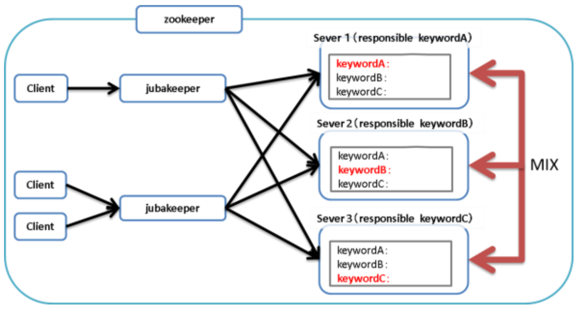

zookeeper
Client
Client
Client
jubakeeper
jubakeeper
Sever 1 (responsible keywordA)
Sever 2 (responsible keywordB)
Sever 3 (responsible keywordC)
keywordA:
keywordB:
keywordC:
MIX

Figure 3. Overview of Burst Detection Framework on Jubauts

In our proposed framework, as shown in Figure 3, each server is in responsible for a set of keywords' burst detections. These keyword sets are not intersected and determined by a global consistent hash table (HT) stored on each server. When a new keyword is required for burst detection monitoring, for example keywordC, it is broadcasted to all the servers and Server3 will take the responsibility for it according to the mapping result of HT on that server. Other servers will have the knowledge that keywordC is monitored on server3, and will transfer client query about keywordC's burst status to Server3 if they don't have that information. The detected burst results on each server will be MIX at some predefined intervals. After the mix, each server may answer all the client queries on any keywords with its local information. The query may still transferred to responsible server if client specifies the freshness of burst results. Note those data transfer amounts during mix are, 'sending': (number of responsible keywords) * windows_length * (size_of_int*2+ size_of_double); 'receiving': (number of all keywords) * windows_length * (size_of_int*2+ size_of_double).

In addition, the mixed burst information on each server provides an opportunity for deeper analysis of the correlated bursts by using the machine learning library contained in Jubatus, such as classification, outlier detection etc. This capability could be very important in extending the usage of burst detection. For example, burst detection could be used as data pre-processing/clean before knowledge mining.

Note that the zookeeper in our framework is in charge of servers (add/delete) management, primary mix server selection and HT maintenance. For the paper length limitation, we omit its description.

5. Experiments

5.1 Experimental environment

Virtual server	Intel Core i7 3.2GHz; Memory 2G
Jubatus Client	1 virtual server
Jubakeeper	1 virtual server
Jubatus Server	1~5 virtual servers

5.2 Experimental configurations and dataset

Keywords number	10, 100, 1000
Window sizes	10, 100, 1000
Document added	1000 documents
Batch size	10 document
HPQ tick data	2013.6.1~2013.11.15; (1745828 trans)

5.3 Experimental results

In our first experiment, we examine the maximum documents adding speed, that all the keywords' burst should have been detected before the next document is added. This speed depends on the documents broadcasting, grep counting and burst calculation time for all the keyword. We only provided a (windows=1000) result in Figure 4 for the space limitation. Results of other smaller window sizes show less document adding time due to the less calculation overhead, but they show the similar trending in performance change, and we found that

1. The CPU overhead increases as the keywords number grows.
2. The CPU overhead reduces as the number of servers grows.
3. The document adding time increases as the number of servers grows (larger communication cost).

In short conclusion, we verified that our proposed system supports real-time burst detection of target-events (keywords) in background-events (document) which happening at msecs time intervals. This interval shall meet the real-time requirement in most burst detection applications. And the interval may be further reduced by scaling out our distributed framework.

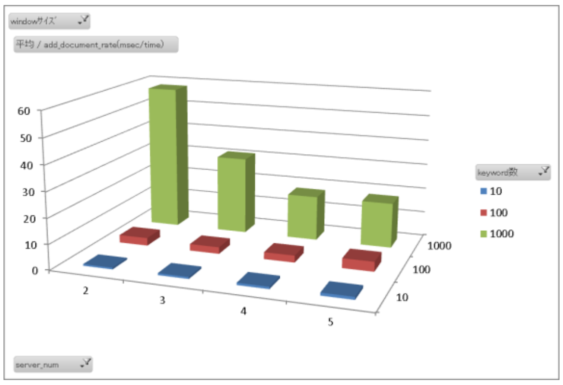


Figure 4. Document adding time (window size =1000)

In our second experiment, we use the proposed system to detect stock price burst. Our first purpose is to verify the detection speed in proposed system fulfills the requirement in real practical usages, such as stock price burst detection. In addition, for deeper stock price analysis, such as similar burst classification, we also verified that using burst weight as price variation indicators leads to higher accuracy in classification result, compared with only using stock price.

In this test, we use one Jubatus server configuration; (windows size = 400). Burst detection is executed after every tick price is added. We found the detection frequency of our system achieves 120 times/sec. As far as we know, this speed overcomes almost all the companies' tick frequency in Nasdaq and NYSE.

We also did stock price variation classification test. In this test, we used a stream data similarity module [6] to classify the most similar price changes. We assume stock's price will change in the same pattern when the same reason repeats. Although this naive assumption is not always the fact, it does appear in some typical cases. For example, we verified that after HP's CEO M. Whitman affirmed the company growth, eg. 2013-06-12-10:45, 2013-10-09-15:14 and 2013-10-10-09:52, HP's shares were all increased over 5%. We found these related burst could be classified with work [6] successfully by using burst degree input, while failed by using the original stock price, as shown Figure 5.

This test results demonstrate that burst detection could also be used as real-time data pre-processing solution, and provide a higher quality of data for deeper analysis.

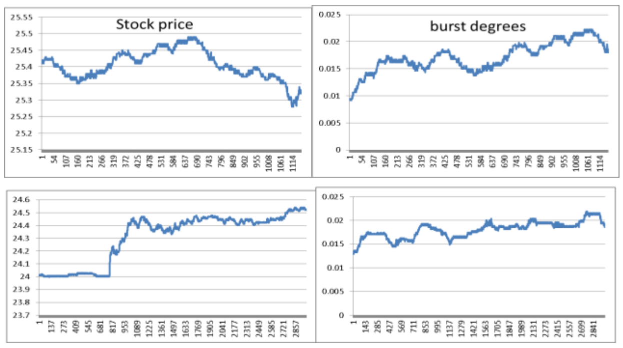


Figure 5. Stock Prices and their Burst Degrees

6. Conclusion

We proposed a real-time burst detection solution based on Kleinberg algorithm, and assessed the scalability and response time of a prototype based on Jubatus. We conclude our system fulfill most of real-time burst detection application requirement.

In near future, we will open our source code under Jubatus project [4], and provide compound usage solutions by combining other machine learning methods in Jubatus.

References

[1] M. Vlachos, K. Wu, S. Chen, and P. S. Yu. Fast burst correlation of financial data. In Proc. of the 9th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'05), Springer-Verlag, Berlin, Heidelberg, pp 368-379.

[2] M. Karnstedt, D. Klan, C. Pölitz, K. Sattler, and C. Franke, Adaptive burst detection in a stream engine. In Proc. of the 2009 ACM Symposium on Applied Computing (SAC '09), pp 1511-1515, New York, NY, USA.

[3] J. M. Kleinberg. Bursty and hierarchical structure in streams. In Proc. of the 8th ACM SIGKDD Intel. Conf. on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada, pages 91-101, ACM, 2002.

[4] http://jubat.us/ja/; (latest accessed on 2014.3.10)

[5] 山西健司:データマイニングによる異常検知, 共立出版

[6] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream Monitoring under the Time Warping Distance. Proc. 23rd Int'l Conf. Data Eng. (ICDE), 2007.