Edge-assisted Detection and Summarization of Key Global Events from Distributed Crowd-sensed Data

Abdulrahman Fahim[†], Ajaya Neupane[†], Evangelos Papalexakis[†], Lance Kaplan[*]
Srikanth V. Krishnamurthy[†], Tarek Abdelzaher[‡]
[†] University of California, Riverside, [*]Army Research Lab
[‡]University of Illinois at Urbana Champaign
afahi002@ucr.edu, ajaya@ucr.edu, epapalex@cs.ucr.edu, lance.m.kaplan.civ@mail.mil
krish@cs.ucr.edu, zaher@illinois.edu

***Abstract*—This paper introduces a novel service for distributed detection and summarization of crowd-sensed events. The work is motivated by the proliferation of microblogging media, such as Twitter, that can be used to detect and describe events in the physical world, such as protests, disasters, or civil unrest. Since crowd-sensed data is likely to be distributed, we consider an architecture, where the data first accumulates across a plurality of edge servers (e.g. cloudlets or repositories) and is then summarized, rather than being shipped directly to its ultimate destination (e.g., in a remote cloud). The architecture allows graceful handling of overload and bandwidth limitations (e.g., in scenarios where capacity is impaired, as the case might be after a disaster).**

When bandwidth is scarce, our service, `BigEye`, only transfers very limited metadata from the distributed edge repositories to the central summarizer and yet supports highly accurate detection and concise summarization of key events of global interest. These summaries can then be sent to consumers (e.g., rescue personnel). Our emulations show that `BigEye` achieves the same precision and recall values in detecting key events as a system where all data is available centrally, while consuming only 1% of the bandwidth needed to transmit all raw data.

I. Introduction

In scenarios like disaster aftermaths, one can visualize crowdsensed data to be dispersed across a set of geographically distributed sources because of strained infrastructure. If all of this data were available at a central entity, it can be analyzed to detect global events of interest. However, in such scenarios it is very common for network infrastructure to be damaged, thereby causing the available bandwidth to be severely constrained. This makes it prohibitive to transfer large volumes of the raw data to such a central entity. For example, during and after the tropical storm Harvey, the failure of several cell towers [41] strained the available bandwidth. If it is possible to somehow only retrieve small amounts of data from the distributed producers, and yet be able to detect events of global significance with high accuracy, it could significantly aid subsequent efforts of search and rescue following the disaster. Our work targets this important problem.

Specifically, our paper describes the benefits of an edge-assisted architecture for crowd-sensing based on social media posts. A plethora of prior work suggests the use of social media for physical event detection [40] (also called crowd-sensing or social sensing in recent literature [13], [3], [43], [40]). Such prior work describes detection algorithms, but assumes that all relevant microblog data is already available in one place. In contrast, this paper considers a distributed architecture, where posts emitted by users are accumulated at the edge then prioritized for collection in a manner that allows clients to retrieve data summaries of user-determined granularity directly from edge servers. Hereafter, we call these edge servers, the *producers*. The paradigm is consistent with cloudlets [34] and distributed proxies [19], [36]. The resulting service, we call `BigEye`, shares very small amounts of metadata between the distributed producers and a central entity (which we call the *summarizer*). Once global events are detected, `BigEye` composes a summary to provide insights for consumers on events of interest. We implement `BigEye` using Twitter as an example, and showcase its benefits.

Since each producer only has a local view of reported events, it is unable to determine, by itself, which events are of global significance. On the other hand, transferring all local data from all producers to the central summarizer is wasteful, especially in cases where the available bandwidth is strained. The first challenge is then to identify key global events while transferring only very limited metadata from each producer to the summarizer. Multiple producers may detect the same global event but there is no easy way to determine that these triggers correspond to the same event. Thus, a second challenge is to reconcile (possibly inconsistent) metadata from multiple producers that correspond to a common event.

In a nutshell, to address these challenges, `BigEye` first allows each producer to individually identify a set of local events that are likely to be of global interest via a measure of what is called the local information gain. Metadata associated with these local events are *pushed* to the summarizer, which then *pulls* additional metadata from a subset of producers as needed. We show formally that the global events detected will be identical to those detected if all the data was available centrally. Once the global events are detected, `BigEye` uses a lightweight method to reconcile common events across a plurality of producers. Finally, using lightweight measurements of bandwidth to the various producers, it can parallelize the transfer of (heavier) content from the producers, such as pictures referred to in tweets, to compose (illustrated) summaries of all events within a very short time. Evaluation shows that the system reduces bandwidth consumption to the summarizer by 99% over approaches that communicate all raw data, while remaining able to detect all key global events that would have been detected if all the data was available centrally.

II. Baseline: Centralized detection

We first provide a description of a baseline method which allows global event detection when all the data are centrally available. The approach is largely based on Storyline [43], which detects new events by identifying new word combinations that occur suddenly with high frequency in a stream of tweets. For example, consider an event, where a drunk driver kills a running dog on a bridge. Tweets that describe the event might include such words as "drunk" and "dog" in the same tweet, generating a burst in the co-occurrence of these two keywords (compared to their co-occurrence rates observed in previous windows). Storyline uses an information gain metric to detect such bursts and filters all tweets that contain the bursting new keyword pair as belonging to the same event.

Formally, the information gain associated with a keyword pair s_z, across time windows $k-1$ and k is given by [43]:

$$infoGain = H(Y) - H(Y|s_z) \tag{1}$$

In the above equation $H(Y)$ and $H(Y|s_z)$ are computed as follows:

$$\begin{aligned} H(Y) = &-\frac{N_k}{N_k + N_{k-1}} \log \frac{N_k}{N_k + N_{k-1}} \\ &-\frac{N_{k-1}}{N_{k-1} + N_k} \log \frac{N_{k-1}}{N_k + N_{k-1}} \quad \text{and,} \end{aligned} \tag{2}$$

$$\begin{aligned} H(Y|s_z) = &-\frac{N_k^z}{N_k^z + N_{k-1}^z} \log \frac{N_k^z}{N_k^z + N_{k-1}^z} \\ &-\frac{N_{k-1}^z}{N_{k-1}^z + N_k^z} \log \frac{N_{k-1}^z}{N_k^z + N_{k-1}^z} \end{aligned} \tag{3}$$

where, N_k is the number of tweets in the current (k^{th}) time window, and N_{k-1} is the number of tweets in the immediately previous time window, $k-1$. N_k^z and N_{k-1}^z are the number of tweets that contain the keyword pair s_z in the current and previous time window, respectively. Note that, these expressions roughly characterize the entropy (conditional entropy) relating to the tweet volume (number of tweets with the specific keyword pair) in consecutive time windows. More details can be found in [43]. Having computed the information gain for the various keyword pairs, a threshold is chosen and all keyword pairs whose information gains are higher than that threshold are considered to have associated physical events that are of interest. Those keyword pairs are denoted as *discriminative pairs*. Prior work also empirically determined good information gain thresholds to use for event detection. In the rest of this paper, we show how to distribute the event detection service.

III. Overview and Assumptions

The BigEye architecture comprises a set of producers, a summarizer and consumers (users). BigEye seeks to identify key events of global interest, occurring at specific time windows of fixed duration (windows also called epochs). A producer P_i ($i \in \{1, m\}$) is an entity that collects sensed data from a local region (e.g., a microblog repository). In BigEye, we assume the time epochs are synchronized across all the producers (we assume that protocols such as NTP can facilitate this [26]); without loss of generality, we denote the index of the time window of interest by k.

The summarizer is a central entity (e.g., a cloud server) that receives appropriate data from all the producers, identifies key events, and composes a summary. We assume that all the producers are connected to the summarizer via a network. The transfer of all local data from the m producers to the central summarizer is considered to be expensive in terms of bandwidth consumption (for example, because of limited bandwidth, congestion or both). Instead, each producer identifies those local events that are likely to be of global interest. It creates appropriate metadata corresponding to these events, and pushes the same to the summarizer. The summarizer constructs a summary which can be used to optionally trigger retrieval of related heavier content (such as web pages, images, or video clips whose URLs appear in the tweets). A consumer or user is connected to the summarizer, and queries for summaries of key events. A depiction of the modules and the composition of BigEye is shown in Fig. 1. In the rest of this paper, we focus on the first two blocks in the figure. The design of the last block (multimedia content retrieval) is a somewhat orthogonal concern and is delegated to a different publication. The key notations used in the paper are summarized in Table I.

IV. Distributed Event Detection

The distinguishing aspect of BigEye (compared to prior approaches and in particular the baseline case described in § II) is that a realistic scenario wherein the sensed data is distributed across multiple geographically dispersed producers, is considered. Blindly transporting all the raw data from these producers to a central entity or summarizer for creating summaries as discussed before is prohibitive. Given this constraint, the question we seek to answer here is "How can we determine which events are of global significance if we don't transfer all the data to the central summarizer?"

The information gain metric (§ II) was computed assuming that the entire data set was available centrally. However, now each producer has a local view and we need an approach to estimate the significance of a local event at the global level (what local events would also be flagged as key events globally if data was centrally available ?). If all the producers were to simply report the "number of tweets" to the summarizer, in the two consecutive time windows of interest (say $k-1$ and k), $H(Y)$ can be computed. However, the challenge lies in computing $H(Y|s_z)$ globally since (a) all keyword pairs and the associated tweets are not known centrally and (b) each producer will only see part of the data and can only compute $H(Y|s_z)$ based on its local dataset. To address these issues, we

TABLE I: Key notation

Symbol	Description
N_k	# of tweets in time window k
N_k^z	# of tweets in time window k that contain the pair s_z
p_s	$\frac{N_k^z}{N_k^z + N_{k-1}^z}$
k	index of the data stream window
$\in$	$H(Y) - threshold$
m	# of producers
i	Producer index
P_i	i^{th} producer in the system
r	ratio of occurrence of keyword pair s_z in time window k to the corresponding occurrence in time window $k-1$

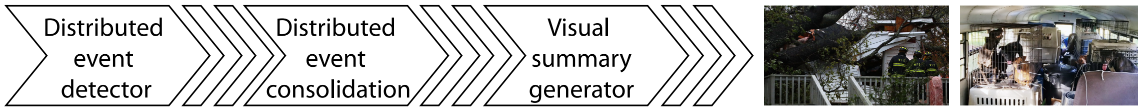

Fig. 1: A high level depiction of `BigEye` with its modules

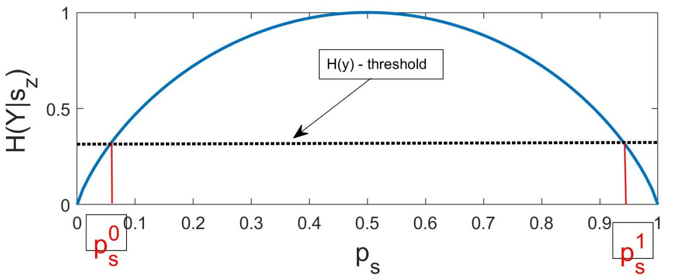

Fig. 2: $H(Y|s)$ with varying p_s. p_s^0 and p_s^1 are two intersecting points with $H(Y) - threshold$

first map a global requirement on information gain (and thus $H(Y|s_z)$) to a local requirement at each individual producer. Later, we reconcile inconsistencies by having the summarizer pull appropriate data from a subset of producers.

What values should the global $H(Y|s_z)$ take to achieve high information gain? Before, we describe our approach in more detail, we first ask the above question. As pointed out above, $H(Y)$ only depends on the number of tweets in consecutive time windows. Thus, the discriminatory term that dictates the information gain with respect to a keyword pair (say s_z) is $H(Y|s_z)$. It is obvious that the lower the value of this term, the higher the information gain associated with the keyword. With reference to Equation 3, let us define $p_s = \frac{N_k^z}{N_k^z + N_{k-1}^z}$. Then, $\frac{N_{k-1}^z}{N_{k-1}^z + N_k^z} = 1 - p_s$. In Fig. 2, we plot $H(Y|s_z)$ as a function of p_s. We see that the lowest values of $H(Y|s_z)$ (yielding the highest information gain) are achieved when p_s is very small or very large (approaches 1). Since p_s corresponds to the probability of having a high number of tweets in frame k relative to the previous frame, it must be large (not small) in order to reflect a new event of interest (otherwise it indicates an event that was of interest in frame $k-1$ but has died down). In other words, the takeaways from the above discussion are (a) $H(Y|s_z)$ must be small (say some small value ϵ) and (b) the corresponding probability p_s as defined above must be large (we require it to be > 0.5).

Let r be *the ratio of occurrence of a keyword pair in the current time epoch to the corresponding occurrence in the previous time epoch*. Let p_s^* be the value of p_s that makes $H(Y|s_z) = \epsilon$. Since we cannot directly obtain p_s^* in closed form by solving $H(Y|s_z) = \epsilon$, we numerically solve it using the Newton method [23] and from among the results, choose the value that is > 0.5. From p_s^*, we compute $\frac{N_k^z}{N_{k-1}^z}$ and denote it as r^*. r^* is the minimum (global) threshold with respect to the ratio of occurrences of a keyword pair in consecutive time epochs, that must be met if the associated keyword pair is to signify an event of interest. In other words, if for a keyword pair $r \geq r^*$, then that keyword pair is a discriminative pair. Next, we provide a formal proof of this claim.

Lemma 1. *Any pair with $p_s \geq p_s^*$ has a ratio of occurrence, r greater than or equal to r^**

Proof:

$$p_s = \frac{N_k^z}{N_k^z + N_{k-1}^z} \tag{4}$$

$$p_s N_k^z - N_k^z = -N_{k-1}^z p_s \tag{5}$$

$$\frac{N_k^z}{N_{k-1}^z} = \frac{p_s}{1-p_s} \tag{6}$$

Note that r is nothing but $\frac{N_k^z}{N_{k-1}^z}$. If we can show that $\frac{p_s}{1-p_s}$ is a non-decreasing function i.e., $\frac{p_s}{1-p_s} \geq \frac{p_s^*}{1-p_s^*}$ if $p_s \geq p_s^*$, then we can infer that $r \geq r^*$ if $p \geq p_s^*$. Let us denote $\frac{p_s}{1-p_s}$ by $F(p_s)$. We show that $F(x) \leq F(y)$ for $0 < x \leq y < 1$. We need to show that

$$\frac{x}{1-x} \leq \frac{y}{1-y} \quad \text{or,} \tag{7}$$

$$x(1-y) \leq y(1-x) \quad \text{or,} \tag{8}$$

$$x - xy \leq y - xy \quad \text{or,} \tag{9}$$

$$x \leq y \tag{10}$$

which is true by assumption. ■

Choosing a local threshold: Given the global threshold, r^*, we need to derive an appropriate local threshold; each producer would estimate r with respect to each keyword pair and if this r is lower than the local threshold one can deem that those keyword pairs are not of global interest. In order to retrieve all the discriminative pairs of global interest (those that would have been detected if all data was available centrally) we need to be conservative i.e., the choice of the local threshold must account for the worst case scenario. By doing so, we can achieve the same precision and recall values with `BigEye`, compared to a centralized baseline (discussed in § II). We point out here, that one may experience an outlier case, where there are no (zero) tweets with a keyword pair in window $(k-1)$ but a significant number in window k; to avoid the divide by zero possibility, we assume that each pair appears at least once at each time window; this fix has almost no influence on the ratios that we are trying to compute.

Given the above threshold and based on the following theorem, we choose the local threshold to be $\frac{r^*}{m}$ if there are m producers.

Theorem 1. *If a keyword pair has a global ratio of occurrence r which is $\geq r^*$, the* local *ratio of occurrence of that keyword pair must be larger than or equal to $\frac{r^*}{m}$ at one or more of the m producers.*

Proof: Let the number of occurrences of a keyword pair in the current and previous time windows be N_k^z and N_{k-1}^z, respectively. The ratio of occurrence of the pair at the global level $(\frac{N_k^z}{N_{k-1}^z}) \geq r^*$.

Case 1: $\frac{N_k^z}{m}$ ***is an integer***:

Let the number of occurrences of the pair in window k (current window) at the local producers be:

$\frac{N_k^z}{m}$ + c_1, $\frac{N_k^z}{m}$ + c_2,..., $\frac{N_k^z}{m}$ + c_m, where c_i and $\frac{N_k^z}{m}$ are integers.

The summation of all the occurrences, across all producers should be equal to the global count N_k^z, i.e., $(\frac{N_k^z}{m}+c_1)+(\frac{N_k^z}{m}+c_2)+...+(\frac{N_k^z}{m}+c_m)=N_k^z$.

Thus, $m\frac{N_k^z}{m}+\sum_i c_i = N_k^z$, hence, $\sum_i c_i = 0$

Case 1A: P_i with $c_i \geq 0$.

The ratio of occurrence of the keyword pair of interest at P_i is $\frac{\frac{N_k^z}{m}+c_i}{N_{k-1}^{z'}}$, where $1 \leq N_{k-1}^{z'} \leq N_{k-1}^z$. Here, $N_{k-1}^{z'}$ is the number of occurrences of the keyword pair in the previous time window (window $(k-1)$) at P_i; naturally this is $\leq$ the global count in that window.

The next step shows that the theorem holds regardless of the value of $N_{k-1}^{z'}$.

$\frac{\frac{N_k^z}{m}+c_i}{N_{k-1}^{z'}} = \frac{N_k^z}{mN_{k-1}^{z'}} + \frac{c_i}{N_{k-1}^{z'}} \geq \frac{N_k^z}{mN_{k-1}^{z'}} \geq \frac{N_k^z}{mN_{k-1}^{z}}$.

But $\frac{N_k^z}{mN_{k-1}^{z}} \geq \frac{r^*}{m}$ and hence, the local rate of occurence at P_i is higher than $\frac{r^*}{m}$.

Case 1B: P_i with $c_i < 0$.

Since $\sum_i c_i = 0$, there there must be at least one other producer with $c_l > 0, l \neq i$; Case 1A will now apply to that producer l.

Case 2: $\frac{N_k^z}{m}$ ***is not an integer***:

If all occurrences at local producers are $\lfloor\frac{N_k^z}{m}\rfloor$, their summation becomes smaller than N_k^z. Hence, the number of occurrences with respect to at least one of the producers, denoted as P_i, must be $\geq \lceil\frac{N_k^z}{m}\rceil$. In other words, the ratio of occurrence at P_i is $\frac{\lceil\frac{N_k^z}{m}\rceil}{N_{k-1}^{z'}}$, where $1 \leq N_{k-1}^{z'} \leq N_{k-1}^z$. Similar to the previous case, $\frac{\lceil\frac{N_k^z}{m}\rceil}{N_{k-1}^{z'}} \geq \frac{r^*}{m}$. ■

Distributed event detection algorithm: Based on the above findings, BigEye applies the following algorithm for distributed event detection.

1) Each producer computes the ratio of occurrences of keyword pairs available locally and transmits the pairs having ratios larger than or equal to $\frac{r^*}{m}$ to the summarizer.
2) The summarizer sends a list of the received pairs to all the producers and inquires about the occurrences of those pairs at the producers. Any producer that had identified that keyword pair, but had not reported it (because it did not meet the threshold) now reports the number of occurrences of that pair. Once this information is available, the summarizer computes the global ratios of all pairs received in step (1).
3) The summarizer filters out pairs with the global ratios less than the global threshold, r^*.

Theorem 1 proved that any globally significant keyword pair "will" be reported by at least a single producer in the first step above. This proves the following lemma.

Lemma 2. BigEye*'s distributed detection algorithm achieves 100% precision and recall, relative to centrally available data.*

Discussion: The performance of our algorithm degrades when the local threshold is very small. When the local threshold becomes very small, the number of keyword pairs sent by the producers to the summarizer increases drastically. Specifically, this happens when the number of producers m is very large or the global threshold r^* is very small, or both. When the local threshold becomes very small (say has a value 1) each producer sends all the pairs; to avoid division by zero we had implicitly set the ratio of occurrence of any pair at a local producer to be greater than or equal to 1. However, in practice, these cases are not of interest. A very large set of producers will imply that the local data consists of small sets, and thus, it will be hard to detect events that are of global interest. A very small threshold will also fail in discriminating between key events of interest and others.

V. Event Consolidation

Different discriminative pairs detected by the summarizer (based on local reporting from the producers) may refer to the same physical event. This is because a single event can be characterized by multiple discriminative keyword pairs. For example, the two keyword pairs (drive, drunk) and (dog, bridge) could refer to a car accident where a drunk driver ran over a running dog on a bridge. It is important to consolidate similar events to avoid redundant summarization i.e., retrievals of unnecessary (redundant or excessive) visual content pertaining to the same event. To aid consolidation, BigEye groups the microblogs containing specific "keyword pairs" into clusters. When two keyword pairs represent the same event, we expect the two corresponding clusters to have similar sets of microblogs and thus consolidate the pairs. Assessing this similarity in the distributed environment is challenging. Naively sending the entire cluster of words associated with a keyword pair to the summarizer for computing a "similarity score" defeats the purpose of reducing communication costs. Thus, BigEye consists of a lightweight approach to consolidate events consisting of the two steps below.

Step I: First, BigEye tries to consolidate the keyword pairs representing similar events at the local producers. In particular, it consolidates events corresponding to "keyword pairs," for which the associated clusters have content that are *very similar*. We use the Jaccard distance [29] to measure the similarity between the two clusters (events). Previous work [43] has reported that the Jaccard distance outperforms other similarity metrics for event consolidation in this way.

Returning to our earlier example, all the tweets about the 'drunk driver who ran over a dog on the bridge' should have the keywords pairs (drunk, driver) and (dog, bridge). Thus, if we compute similarity score between two local clusters corresponding to the keyword pairs (drunk, driver) and (dog,bridge), one can expect the score is very high (in our experiments we find such scores to be in line with what is computed over all the tweets when available centrally). Based on this, we compute the similarity between events represented by keyword pairs at local producers. If the distance between these two events is above the given threshold, the local producer sends the keyword pairs representing these events to the summarizer reporting that they should be consolidated. At the summarizer, if the majority of the producers ($\geq 50\%$) indicate that two keyword pairs should be consolidated, the summarizer sends feedback to all the producers to merge the contents associated with these keyword pairs. This helps to consolidate highly similar events at individual producers, without sending the entire cluster contents to the summarizer.

Step II: In the second step, BigEye further tries to consolidate global events that do not have very high similarity locally at individual producers, and were not consolidated in Step I. To achieve this objective, it seeks to only exchange minimal information with the summarizer to limit bandwidth consumption. Specifically, it employs minHash [9] and Locality Sensitive Hashing (LSH) [15] functions at each producer, to convert a cluster of words represented by a discriminative keyword pair into a set of hash integers. minHash and Locality Sensitive Hashing (LSH) are techniques commonly used to measure the similarity of large documents within reasonable running times ([15], [21]). The probability that hashes of two sets are similar is equivalent to the corresponding Jaccard similarity of the same sets[5], [21]. Each producer transmits the computed hash values to the summarizer. The summarizer compares the hash values across clusters to measure the similarity globally. The bandwidth consumed on sharing the value generated by minHash is significantly smaller than the bandwidth consumed on sharing the entire cluster to the summarizer (as will be shown in § VI). At this point, BigEye has tried to reconcile the possibility that a single global event of interest was perhaps identified as different events because there were multiple keyword pairs from tweets that were used as discriminatory features for this event. While we are not able to completely eliminate a single event being wrongly classified as multiple events, this process drastically reduces the possibility.

VI. Implementation and Evaluation

In this section, we describe in detail our experimentation environment and evaluate the performance of each module of BigEye. We implement BigEye on the top of the popular Mininet[27], a software defined network (SDN) emulator. Specifically, we use the NDN test-bed topology [28] which can be considered as the representation of a real scenario with distributed producers. We use the default latencies per link unless specified otherwise; in some cases, we alter the link capacities to showcase specific results. Note that we do not use name-based forwarding in our work; rather, we use default IP based routing methods (this is what is used with smartphones or social media transfers today).

We have producer nodes in the network that stream microblog data, and a summarizer that coordinates with these producers in the three stages of BigEye, and finally composes summaries. We used the ONOS SDN controller [32] to manage the flows from the producers to the summarizer (and vice versa). We use the default settings in ONOS [7]. Python was used in the implementation of all BigEye modules.

A. Datasets and Distribution

We collect two datasets of tweets, one of which is related to protests within a time frame (Protest), and the second to the hurricane Florence (Florence), using the Twitter API. We clean up the dataset by removing re-tweets, stop words [33], special characters, links and attachments, and do stemming [31]. For that purpose, we use Python-NLTK tokenizer [38] and Porter stemmer [31]. Below we summarize the datasets collected:

- **Protest.** The dataset was collected from *March 18, 2018* to *April 18, 2018* with keyword "protest". It consists of approximately 300,000 tweets after filtering out the retweets.
- **Florence.** The dataset was collected from *Sept 14, 2018* to *Sept 24, 2018* with keywords "hurricane" and "florence". It contains approximately 100,000 tweets after filtering out the retweets.

We stream the data as feeds to the producers to emulate a real-time situation. We choose the window size to be 24 hours. We use the term *instance* to refer to the data corresponding to each such window size. Thus, the total number of instances considered (including Protest and Florence datasets) in our paper is 40. For distributing the data over multiple producers, we consider two different scenarios detailed next.

1) Natural distribution: In practice, the tweets are posted by Twitter users from different geographical locations. However, the geolocation of the tweet is included in only less than 2% of Twitter data[10] which makes simulating the exact real-world geographical distribution of tweets difficult. So, in this paper, we used users' addresses (available via users' profiles) to simulate the geographical distribution of tweets across multiple producers. Even then, some of the users' had provided vague addresses (e.g., "on the floor" and "sky"). We could only retrieve the location of 60% of users in Protest and 70% of users in Florence Hurricane datasets. We converted these addresses to geolocations (latitude and longitude) using the API provided by *HERE.com* [16].

Recall that the considered NDN topology reflects real server locations; it is assumed that each of these servers is a producer and each tweet is sent to the nearest producer. An example is shown in Fig. 3. The model can be extended to capture a variable number of producers that is less than the total number of producers in the NDN-topology as follows. We first send each tweet to the nearest producer (all nodes in the network are potential candidates for being a producer) and then select the top x locations where x is the selected number of producers. For the remaining producers, the tweets are moved to the nearest producer from among the selected x producers. The tweets with no user location are sent to randomly selected producers from this set.

2) Synthetic distribution To further evaluate the performance of BigEye, we consider a synthetic distribution of

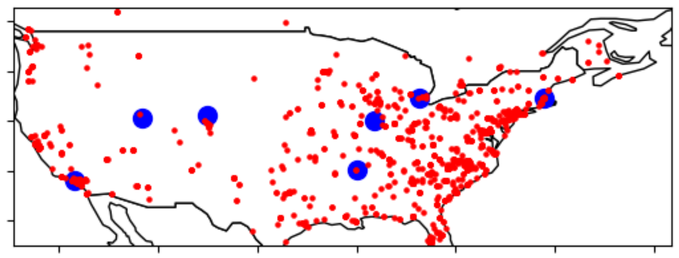

Fig. 3: A united stated map of two hour tweets of Florence dataset where red dots are sources of tweets and blue dots are the producers. The tweets from each red dot are sent to the nearest blue dot.

tweets across multiple producers. Specifically, the distribution of the data over the producers follow a Gaussian distribution with $\mu = \frac{N_k}{m}$ and $\sigma^2 = \beta * \mu$; we vary the β to control the skew.

B. Evaluation Parameters

We evaluate the performance of BigEye with respect to the case where all data is available centrally. We consider the datasets described in the previous section. Each dataset is streamed separately to adhere to our assumption (we consider streamed data to belong to a particular scope) in § III. The obtained results are then merged before presentation for two reasons. First, we observe that we get consistent performance with BigEye with both datasets when the data is distributed over multiple producers, compared to the case where data is available centrally. Second, BigEye is generic in that it works with any crowd-sensed dataset belonging to a particular scope. Because of these two reasons, showcasing the performance of each dataset independently does not provide any new information but consumes space.

We perform experiments with all the instances from the datasets (40 instances) considered one at a time. To gain statistical significance, we repeat each experiment 50 times. While BigEye was built holistically with all of its three components, we evaluate each component independently to showcase its merits. Towards this, we use the following set ups.

Distributed Event Detection Module: As shown in § IV, our approach always provides the same precision and recall in detecting key events, as that of a system where all data is available centrally. We have validated this experimentally and omit showing the event detection accuracy in the interest of space.

To evaluate if the number of keyword pairs that are returned by BigEye to the summarizer is reasonable, we compare the number with the best case scenario. In particular, we consider an *oracle* that does a brute force search, considering all possible subsets of the keywords pairs sent (the ranked orders of those pairs are still maintained), and checks if any of those subsets yields the same precision and recall as our approach (and the central approach). We choose the smallest subset among these as the best possible scenario (we label it as oracle prediction in the results that we present). In other words, instead of choosing a threshold $\frac{r^*}{m}$, we find a the smallest value $l \leq m$ such that choosing $\frac{r*}{l}$ results in the detection of all events of interest.

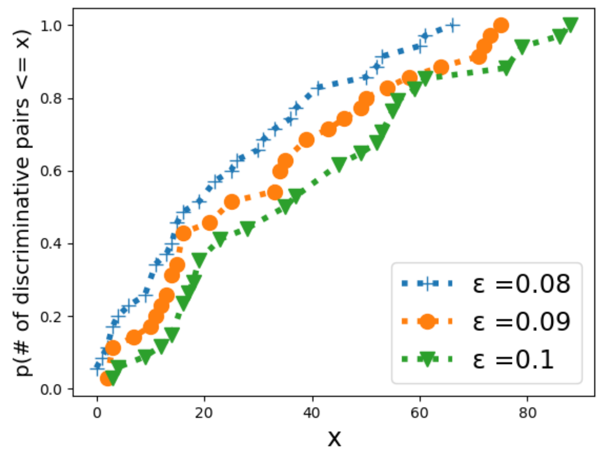

Fig. 4: CDF of the number of detected discriminative pairs with different global thresholds.

Event Consolidation: Next, we assess the performance of our proposed distributed consolidation. Towards this, we compare the BigEye's approach with the case where we send all the data associated with the clusters to the summarizer for consolidation. This baseline is denoted as *central consolidation*. This also corresponds to the consolidation used in Storyline [43]; here the data is available centrally and Jaccard distance is used for similarity assessment.

The metrics of interest are accuracy (defined next) and the amount of data sent from the producers to summarizer for the purposes of consolidation. Accuracy is defined to be the ratio of the number of keyword pairs that are grouped correctly (the events are correctly consolidated) to the total number of keyword pairs. Two keyword pairs are incorrectly grouped if (a) these pairs belong to the same event but are put in different groups and (b) if they belong to different groups (events) but are consolidated into the same group. We also compare the amount of sent data from the producers to summarizer with our approach (in bytes), with the other approach.

C. Results on distributed event detection

First, we evaluate our distributed event detection module using the aforementioned metrics. We recall our discussion in § IV ($H(Y|s_z)$ was to be a small value ϵ), and select ϵ to be 0.08, 0.09 and 0.1 ($r^* = 100,87$ and 76, respectively). Here, we also refer the reader to Table I since the notation therein is used in the discussion.

Effect of r^* on the total number of pairs retrieved by the summarizer. We plot the CDFs of the number of events (corresponding to identified discriminative key word pairs) detected with each value of r^* in Fig. 4. As one might expect, as r^* increases, the number of pairs retrieved decreases (with higher r^* only the most significant events are detected). This effect is also seen in Fig. 5.

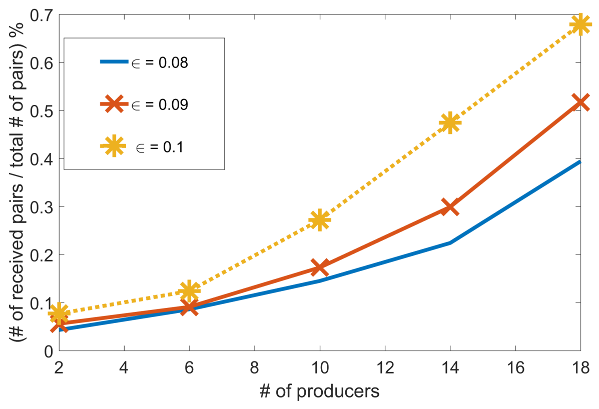

Fig. 5: The performance of our distributed event detection with varying number of producers

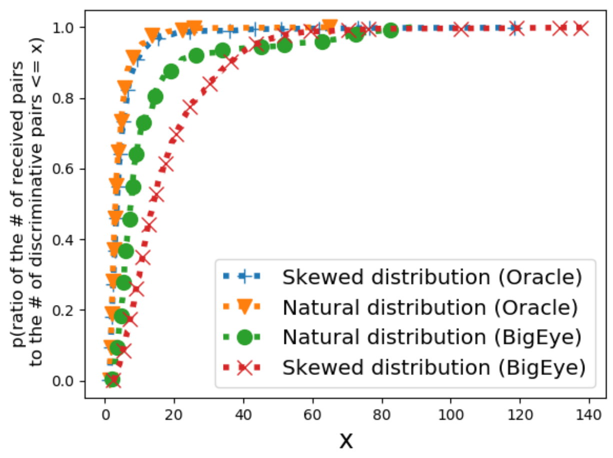

Fig. 6: Performance of our distributed event detection with different data distributions

Effect of increasing number of producers. In Figure 5, we plot the ratio of the number of retrieved keyword pairs to the total number of keyword pairs identified, versus the number of producers. We assume that the data are distributed as per the *natural* distribution. As one might expect, the number of key-words pairs returned to the summarizer increases when as the number of producers, m, increases. This is because $\frac{r^*}{m}$ decreases i.e., a lower or more conservative (local) threshold is used at each of the producers. It is worth noting that with small $r^* = 76, (H(Y|s_z) = 0.1)$ and large $m = 18$, the total number of received pairs is less than 1% of the total number of keyword pairs considered globally.

Comparison with oracle. Next we examine how the number of keyword pairs retrieved with BigEye compares to what is obtained by an oracle, when the data is spread across the producers as per the different data distributions (discussed in subsection VI-A). For the skewed distribution, we choose $\mu = \frac{N_k}{m}$ and $\sigma^2 = 0.5 * \mu$; this ensures a high skew. We fix $\epsilon = 0.09$ (r^*=87), and m to be 10.

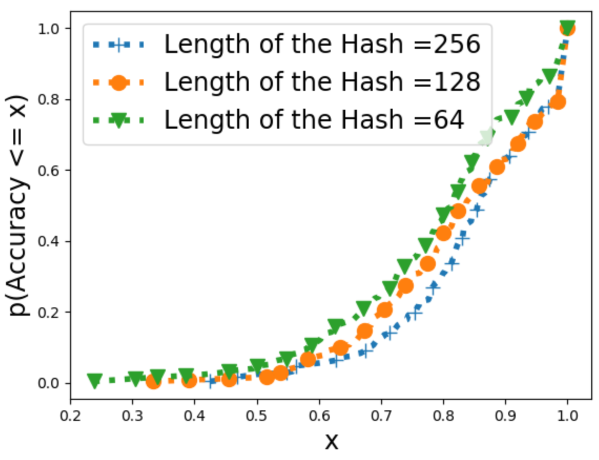

Fig. 7: Distributed consolidation accuray with respect to centralized consolidation (Storyline [43])

.

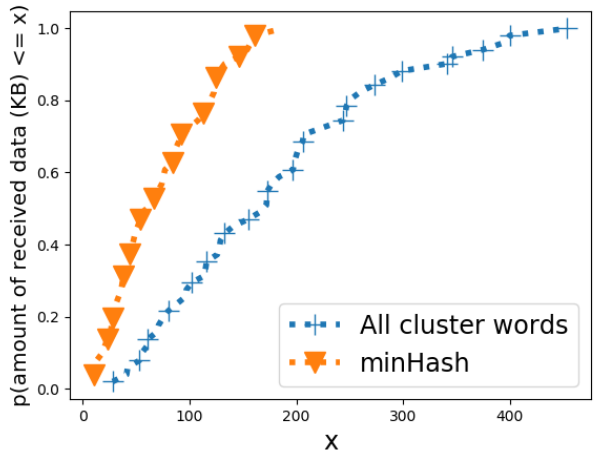

Fig. 8: Bandwidth savings from BigEye's distributed consolidation approach in terms of total amount of data sent from producers to summarizer.

In Figure 6, we plot the CDF of the ratio of the number of pairs received at the summarizer to the number of global discriminative pairs with both BigEye and the oracle based approach described earlier. We see that the performance of BigEye is similar to that of the *oracle* when data is dispersed as per the natural distribution. However, when the distribution is skewed, the performance of the BigEye degrades compared to the oracle. This is because the producers with large numbers of tweets have a large number of keyword pairs that pass the conservative threshold selected by BigEye; thus, they end up sending a large number of pairs that are not useful in detecting key events.

D. Results with regards to consolidation

Next, we evaluate the benefits from BigEye's consolidation module. In our evaluations we use the same ϵ values mentioned earlier in § VI-C. Recall that BigEye consolidates events over two steps. In the first step, we consider a similarity requirement of 0.99 (Jacquard distance) to consolidate events at individual producers. For the second step, we consider three minHash signatures of length 64, 128, and 256 integers. We consider different minHash signatures as it has been reported that the length of generated minHash signatures affects the similarity scores [21]. We also vary the consolidation thresholds that are used centrally from 0.4 to 0.9 with a stepsize of 0.1.

In Figure 7, we compare BigEye's consolidation approach with different minHash lengths. We observe that the similarity estimation improves as the length of the minHash signature increases (less collisions). Similar phenomena have also been reported in previous literature [21]. We observe that, for a minHash signature of length 128, almost 85% of instances show a consolidation accuracy of larger than 70% of what is achieved if all data was available centrally. BigEye provides consistent consolidation accuracies irrespective of the number of producers.

We show next the consolidation benefits in terms of the reduction in the volume of data sent from producers to summarizer in bytes, compared to sending all keywords in the clusters (needed for central consolidation); Fig. 8 shows that our approach reduces the communication costs significantly compared to that baseline approach, and in particular the average cost by 60% (fewer bytes).

E. An Output Example of BigEye

We show in Table II the holistic output of BigEye based on two randomly selected instance from Florence. Specifically, we show the detected discriminative pairs, the textual summaries associated with the detected events and a visual summary corresponding to the discriminative pair. In this implementation, the visual summary is simply a randomly selected image from one of the tweets that belong to the content relating to the summary. To elaborate on these instances, on Sept 15 and 16, two major events relating to saving pets were detected. First a pedestrian was able to free six dogs that were locked inside a cage and abandoned by their owner during the hurricane. The second event related to a truck driver saving 64 dogs during the hurricane. As shown in the table, BigEye distinguishes the two events despite the strong correlation between them (saving pets).

VII. Discussion

As alluded to in § III and Fig. 1, BigEye contains a third module which retrieves appropriate richer image/video content from the producers to form informative visual summaries. In Table II, the visual summary was a randomly chosen image corresponding to the events of interest. In practice however, one may want to retrieve more appropriate content (e.g., the most popular, images with the highest resolution or camera orientation) so as to be more beneficial to the search and rescue responders. Since the data is distributed across a pluarlity of producers, "how to determine the most appropriate content?" is not an easy question to answer. In addition, to ensure that the visual summaries are quickly composed, it becomes important to parallelize the retrieval of visual content from multiple producers to the extent possible (to minimize the time of retrieval). While we have partially addressed these problems, given the space limitations, we delegate the discussion of the approaches we design to a different publication.

VIII. Related work

Recent literature makes an analogy between posts on social networks and sensor data [41], [42]. Similar to BigEye, there are prior studies on detecting events from such sensor data. Allan et al.[4] used "term frequency" (tf) and "inverse document frequency" (idf) features to build a query representation for content from news stories and identified an event when the similarity score of new news story was less than a given threshold in comparison to any previous news query in memory. Similarly, Shamma et al. [35] used a normalized term frequency to identify peaky topics, the terms which are particular to a time window, to detect highly localized events of interest. Benhardus et al. [6] also used tf-idf analysis and relative normalized term frequency analysis on twitter documents to identify trending topics. However, these approaches were reported to be inefficient in separating individual event instances [43]. Moreover, unlike tf-idf, BigEye works by only computing information gain over two consecutive time windows.

Text stream clustering has also been applied for event detection. Ordonez et al. [30], Zhong et al. [45] and Aggarwal and Yu [2], used optimizations of k-means algorithms to cluster data streams for events detection. Similarly, communication patterns [11], social network topological features [1], language specific features [12], [37], [44], and location of tweets [8], [22], [39] have also been used by researchers for clustering data to detect events. Nevertheless, precisely defining the number of clusters (k) for online streaming data is not feasible. Researchers have also used topic modeling for event detection [20], [17], [46]. However, topic based approaches have been reported to be inefficient in identifying events happening in parallel instances [43]. Unlike these methods, BigEye detects events by measuring the temporal bursts in the word-pairs that do not co-occur frequently. BigEye's event detection approach is closely related to Storyline that was proposed by Wang et al. [43]. However, unlike BigEye, Wang et al. only focuses on event detection when data is centrally located.

BigEye centers around the detection of global events by only sharing minimal amounts of information between distributed producers and a central summarizer. There have been some prior work on selectively sending information to a central entity [18], [24], [14]. However, unlike BigEye, these approaches do not focus on event detection. Closely relevant to our study is the study by McCreadie et al. [25]. Unlike BigEye, they do not consider bandwidth constraints and only try to minimize the event detection time by distributing the computational costs of processing documents across multiple machines.

IX. Conclusions

In this paper, we design and implement a framework BigEye that facilitates (a) the detection of key global events

TABLE II: Florence Hurricane Summarization

Date	Detected keywords	Textual summary	Visual summary
Sept 15	sympathy, hurricane, deepest, hurricane	”My deepest thoughts and prayers are with those in North and South Carolina, and Virginia affected by Hurricane Florence” My Deepest Condolences to the families of those that died in Tropical Storm/Hurricane Florence.	
Sept 16	’abandon’, ’hero’, ’six’, ’dog’, ’64’, ’rescue’, ’hurricane’	“Six dogs have been rescued from rising flood waters, after they were locked in a cage and abandoned by their owners” “As Florence loomed, a pet lover escaped South Carolina with 64 dogs and cats on a school bus”	

based on crowd-sensed data that is distributed across geographically spread out producers and (b) the composition of concise summaries that provide a zoomed-in view about such events. A distinguishing aspect of `BigEye` is that it is extremely lightweight in terms of bandwidth consumption i.e., it requires the transfer of very little of the raw crowd-sensed data from the producers to a central entity for both event detection and the following summarization. In spite of this, it is able ot achieve 100 % precision and recall values compared to a case where all the crowd-sensed data is available centrally, while transmitting only 1 % of the crowd-sensed data.

Acknowledgment: This work was partially supported by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also partially supported by the NSF CPS grant 1544969.

References

[1] C. C. Aggarwal and K. Subbian, “Event detection in social streams,” in *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 2012, pp. 624–635.

[2] C. C. Aggarwal and P. S. Yu, “A framework for clustering massive text and categorical data streams,” in *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM, 2006, pp. 479–483.

[3] M. I. Ali, N. Ono, M. Kaysar, Z. U. Shamszaman, T.-L. Pham, F. Gao, K. Griffin, and A. Mileo, “Real-time data analytics and event detection for iot-enabled communication systems,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 42, pp. 19–37, 2017.

[4] J. Allan, R. Papka, and V. Lavrenko, “On-line new event detection and tracking,” in *ACM SIGIR Forum*, vol. 51, no. 2. ACM, 2017, pp. 185–193.

[5] A. Arasu, V. Ganti, and R. Kaushik, “Efficient exact set-similarity joins,” in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 918–929.

[6] J. Benhardus and J. Kalita, “Streaming trend detection in twitter,” *International Journal of Web Based Communities*, vol. 9, no. 1, pp. 122–139, 2013.

[7] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, “Scalability of onos reactive forwarding applications in isp networks,” *Computer Communications*, vol. 102, pp. 130–138, 2017.

[8] A. Boettcher and D. Lee, “Eventradar: A real-time local event detection scheme using twitter stream,” in *2012 IEEE International Conference on Green Computing and Communications*. IEEE, 2012, pp. 358–367.

[9] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, “Min-wise independent permutations,” *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.

[10] Z. Cheng, J. Caverlee, and K. Lee, “You are where you tweet: a content-based approach to geo-locating twitter users,” in *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010, pp. 759–768.

[11] F. Chierichetti, J. M. Kleinberg, R. Kumar, M. Mahdian, and S. Pandey, “Event detection via communication pattern analysis.” in *Proceedings of ICWSM*, 2014.

[12] P. Giridhar, S. Wang, T. F. Abdelzaher, J. George, L. Kaplan, and R. Ganti, “Joint localization of events and sources in social networks,” in *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*. IEEE, 2015, pp. 179–188.

[13] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, “Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm,” *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 7, 2015.

[14] H. Gupta, V. Navda, S. Das, and V. Chowdhary, “Efficient gathering of correlated data in sensor networks,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 1, p. 4, 2008.

[15] T. Haveliwala, A. Gionis, and P. Indyk, “Scalable techniques for clustering the web,” 2000.

[16] Here.com, *Location data processing*, (accessed June, 2018). [Online]. Available: https://www.here.com/

[17] Y. Hu, A. John, D. D. Seligmann, and F. Wang, “What were the tweets about? topical associations between public events and twitter feeds.” in *Proceedings of ICWSM*, 2012.

[18] K. Khalil, A. Aqil, S. V. Krishnamurthy, T. Abdelzaher, and L. Kaplan, “Nest: Efficient transport of data summaries over named data networks.”

[19] D. C. Knowledge., “Twitter Adding More Data Center Space (Again),” https://www.datacenterknowledge.com/archives/2011/09/19/twitter-adding-more-data-center-space-again, Sept 19, 2011, [Online; accessed Oct-11-2018].

[20] J. H. Lau, N. Collier, and T. Baldwin, “On-line trend analysis with topic models:\# twitter trends detection topic model online,” *Proceedings of COLING 2012*, pp. 1519–1534, 2012.

[21] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Finding Similar Items*, 2nd ed. Cambridge University Press, 2014, p. 68–122.

[22] C. Li, A. Sun, and A. Datta, “Twevent: segment-based event detection

from tweets," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 155–164.

[23] D. G. Luenberger, Y. Ye *et al.*, *Linear and nonlinear programming*. Springer, vol. 2.

[24] Y. Ma, Y. Guo, X. Tian, and M. Ghanem, "Distributed clustering-based aggregation algorithm for spatial correlated sensor networks."

[25] R. McCreadie, C. Macdonald, I. Ounis, M. Osborne, and S. Petrovic, "Scalable distributed event detection for twitter," in *Big Data, 2013 IEEE International Conference on*. IEEE, 2013, pp. 543–549.

[26] D. Mills *et al.*, "Network time protocol," RFC 958, M/A-COM Linkabit, Tech. Rep., 1985.

[27] Mininet, *Mininet.org*, (accessed June, 2018). [Online]. Available: http://mininet.org/

[28] NDN-testbed, *Named Data Networking*, (accessed June, 2018). [Online]. Available: https://named-data.net/ndn-testbed/

[29] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, no. 6, 2013.

[30] C. Ordonez, "Clustering binary data streams with k-means," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM, 2003, pp. 12–19.

[31] M. Porter., "Porter Stemmer," https://tartarus.org/martin/PorterStemmer/, Jan, 2006, [Online; accessed Oct-11-2018].

[32] O. project, *ONOS SDN*, (accessed June, 2018). [Online]. Available: https://onosproject.org/

[33] A. Rajaraman and J. D. Ullman, *Data Mining*. Cambridge University Press, 2011, p. 1–17.

[34] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[35] D. A. Shamma, L. Kennedy, and E. F. Churchill, "Peaks and persistence: modeling the shape of microblog conversations," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. ACM, 2011, pp. 355–358.

[36] Telegraph., "Moscow says Twitter ready to store data of users on Russian servers despite concerns over surveillance ," https://www.telegraph.co.uk/news/2017/11/08/moscow-says-twitter-ready-store-data-users-russian-servers-despite/, Nov, 2017, [Online; accessed Oct-11-2018].

[37] I. Tien, A. Musaev, D. Benas, A. Ghadi, S. Goodman, and C. Pu, "Detection of damage and failure events of critical public infrastructure using social sensor big data," in *IoTBD*, 2016.

[38] N. Tokenizer., "NLTK 3.3 documentation," https://www.nltk.org/api/nltk.tokenize.html, [Online; accessed Oct-11-2018].

[39] M. Walther and M. Kaisser, "Geo-spatial event detection in the twitter stream," in *European conference on information retrieval*. Springer, 2013, pp. 356–367.

[40] D. Wang, T. Abdelzaher, and L. Kaplan, *Social sensing: building reliable systems on unreliable data*. Morgan Kaufmann, 2015.

[41] D. Wang, M. T. Amin, S. Li, T. Abdelzaher, L. Kaplan, S. Gu, C. Pan, H. Liu, C. C. Aggarwal, R. Ganti *et al.*, "Using humans as sensors: an estimation-theoretic perspective," in *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*. IEEE, 2014, pp. 35–46.

[42] D. Wang, L. Kaplan, H. Le, and T. Abdelzaher, "On truth discovery in social sensing: A maximum likelihood estimation approach," in *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM, 2012, pp. 233–244.

[43] S. Wang, P. Giridhar, H. Wang, L. Kaplan, T. Pham, A. Yener, and T. Abdelzaher, "Storyline: Unsupervised geo-event demultiplexing in social spaces without location information," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*. ACM, 2017, pp. 83–93.

[44] K. Watanabe, M. Ochi, M. Okabe, and R. Onai, "Jasmine: a real-time local-event detection system based on geolocation information propagated to microblogs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2541–2544.

[45] S. Zhong, "Efficient streaming text clustering," *Neural Networks*, vol. 18, no. 5-6, pp. 790–798, 2005.

[46] X. Zhou and L. Chen, "Event detection over twitter social media streams," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 23, no. 3, pp. 381–400, 2014.