Scaling DBSCAN-like Algorithms for Event Detection Systems in Twitter

Joan Capdevila*, Gonzalo Pericacho, Jordi Torres**, and Jesús Cerquides***

Department of Computer Architecture, Polytechnical University of Catalonia (UPC)
Department of Computer Science, Barcelona Supercomputing Center (BSC-CNS)
Artificial Intelligence Research Intstitute (IIIA-CSIC)
{jc@ac.upc.edu,gonzalo.pericacho@est.fib.upc.edu
jordi.torres@bsc.es,cerquide@iaaa.csic.es}

Abstract. The increasing use of mobile social networks has lately transformed news media. Real-world events are nowadays reported in social networks much faster than in traditional channels. As a result, the autonomous detection of events from networks like Twitter has gained lot of interest in both research and media groups. DBSCAN-like algorithms constitute a well-known clustering approach to retrospective event detection. However, scaling such algorithms to geographically large regions and temporarily long periods present two major shortcomings. First, detecting real-world events from the vast amount of tweets cannot be performed anymore in a single machine. Second, the tweeting activity varies a lot within these broad space-time regions limiting the use of global parameters. Against this background, we propose to scale DBSCAN-like event detection techniques by parallelizing and distributing them through a novel density-aware MapReduce scheme. The proposed scheme partitions tweet data as per its spatial and temporal features and tailors local DBSCAN parameters to local tweet densities. We implement the scheme in Apache Spark and evaluate its performance in a dataset composed of geo-located tweets in the Iberian peninsula during the course of several football matches. The results pointed out to the benefits of our proposal against other state-of-the-art techniques in terms of speed-up and detection accuracy.

Keywords: Event detection, parallel algorithm, data clustering, DBSCAN, MapReduce, Apache Spark, Twitter

1 Introduction

Event detection seeks to identify and characterize anomalous patterns in data which are typically caused by some real-world phenomena [1]. For example, authors in [2] aimed to detect space-time clusters in a dataset composed of brain

* Obra Social "la Caixa".
** Spanish Ministry of Economy and Competitivity under contract TIN2015-65316 and BSC-CNS Severo Ochoa programs (SEV2015-0493, SEV-2011-00067).
*** The SGR program (2014-SGR-1051) of the Catalan Government and the COR (TIN2012-38876-C02-01) project.

cancer cases in Los Alamos, New Mexico during 1973 - 1991. From the discovered clusters, their goal was to find whether these clusters occurred by chance or due to some real-world cause such as the presence of Los Alamos National Laboratory, a nuclear research and design facility.

Recently, the increasing use of social networks with location services has converted social network users into actual sensors capable of ubiquitously reporting real-world events [3]. These virtual communities enable the simultaneous identification of various types of events ranging from natural disasters [4] to geo-social events [5]. Particularly, Twitter[1] has shown to be more effective and faster than traditional media channels. For example, in reporting Osama Bin Laden death [6] or Mumbai attacks [7].

However, event detection in Twitter poses a set of new challenges [8]. In contrast to classical fields of application, Twitter contains tones of non-event observations such as *memes*, user conversations or *retweets*, making it very hard to uncover newsworthy events [9]. Hence, event detection techniques need to explicitly distinguish between event and non-event tweets in order to uncover these hidden patterns. Furthermore, more than 500 millions tweets are generated worldwide per day[2], entailing a high computational cost to process this huge amount of data in a single machine. Therefore, the parallelization and distribution of such algorithms plays a key role to design and implement practical event detection systems on a national or worldwide scale.

A *bottom-up* approach to retrospectively detect events from spatio-temporal data such as geo-located tweets is based on DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [10]. This clustering algorithm is well-known for its noise resilience capability which enables to handle non-event observations in Twitter. Authors in [11, 12] proposed to use the spatio-temporal extension of DBSCAN called ST-DBSCAN [13] to detect specified events (i.e. precipitation and dengue) from text-filtered tweets. Others [14] extended ST-DBSCAN to also consider textual features through the cosine similarity of their term vectors in order to discover various types of unspecified events. Lately, Capdevila et al. [15, 16] presented Tweet-SCAN which also considers text features but it instead relies on the Jensen-Shannon distance over probabilistic topic distributions [17] to search among text.

However, the above-mentioned DBSCAN-like techniques were not initially designed to work in geographically large regions and temporarily long periods with lots of observations. On the one hand, the large amount of tweets, n, directly affects the computational cost of DBSCAN which has an average time complexity of $O(n \log n)$. On the other hand, tweets are spread unevenly over large space-time regions and DBSCAN fails to cluster uneven distributions of tweets, compromising the overall detection accuracy of event detection systems. This uneven distributions are due to the fact that spatial and temporal distributions of tweets are strongly correlated with the activity of the underlying

[1] www.twitter.com
[2] https://blog.twitter.com/2013/new-tweets-per-second-record-and-how

population [18]. Thus, urban areas during peak hours are likely to generate much more tweets than rural areas during off-peak hours.

In this work, we tackle the scaling of DBSCAN-like event detection algorithms, such as Tweet-SCAN [15, 16], for large spatio-temporal regions. Given that the length of tweet messages is limited to 140-character by Twitter, we only focus on the scaling of spatio-temporal dimensions, although we note that text disambiguation is essential in event identification and future work should take it into account. As a result of this, we propose a novel density-aware MapReduce scheme implemented in Apache Spark [19] that parallelizes and distributes DBSCAN-like algorithms to scale event detection in Twitter. In particular, we propose an Octree-based method to partition tweets as per its space-time features. Given that these partitions correspond to similarly dense regions, we introduce a MapReduce scheme that computes local DBSCANs for each data partition with its parameters tailored to the local tweet density. Furthermore, our proposal includes a framework to setup these local DBSCAN parameters so that the overall detection performance is optimized. Last, we provide empirical evidence that this scheme scales well to large data sets and is able to detect events in low and high density regions.

The structure of the remaining sections is as follows. In section 2, we introduce the necessary background regarding DBSCAN algorithms and their parallelization through MapReduce. In section 3, we propose the novel density-aware MapReduce scheme to scale event detection on large datasets over large regions. Then, we present in section 4 the empirical results of our proposal. Finally, we list several conclusions from this work in section 5 and point out future work in section 6.

2 Background

DBSCAN-like algorithms constitute a common *bottom-up* approach to the event detection problem [1]. Within this approach, events are defined as groups of points, a.k.a. clusters, whose point density is abnormally high. DBSCAN [10] defines a greedy algorithm through which points are associated to events. Points which are not assigned to any event are considered noise by DBSCAN. Moreover, DBSCAN does not require to specify the number and shape of events. These features make DBSCAN a suitable framework for event detection in contrast to other popular clustering techniques, such as K-Means.

2.1 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [10] clusters points that are closely together and marks as noise those that are in low-density regions.

DBSCAN is formalized through the following definitions with respect to its parameters (ϵ and $MinPts$) and a dataset of points DB.

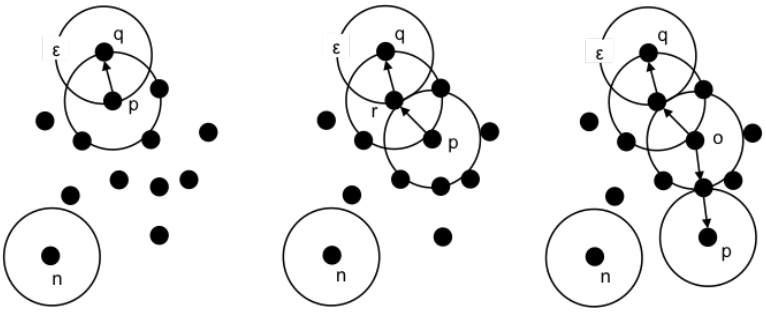


Fig. 1: (left) q is directly density-reachable from p, (middle) q is density-reachable from p and (right) q is density-connected from p

- The ϵ-neighborhood of a point p is the set of points whose distance to p is less or equal than ϵ (see ϵ-circle in Fig. 1).
- A point p is a **core point** if the number of neighbors, in its ϵ-neighborhood, is greater or equal than $MinPts$ (see left Fig. 1 for $MinPts = 4$).
- Given two points p and q, if p is a core point and q belongs to the neighborhood of p, then q is *directly density-reachable* from p (see left Fig. 1).
- q is *density-reachable* from another point p if there is sequence of points $p, r_1, r_2, ..., r_n, q$ such that each point (r_{i+1}) is *directly density-reachable* from the previous r_i (see middle Fig. 1).
- p and q are *density-connected* if there is a point o such that both are *density-reachable* from r (see right Fig. 1).
- A non-empty subset C of DB is a **cluster** if satisfies: (Maximality) For any point $p \in C$ from which q is *density-reachable*, $q \in C$. (Connectivity) For any set of point $p, q \in C$, p is *density-connected* to q.
- A point n is a **noise point** if it does not belong to any cluster (see Fig. 1).
- A point q is a **border point** if it belongs to a cluster but it is not a core point (see left Fig. 1).

The greedy algorithm defined in DBSCAN uncovers clusters of points following on the above definitions. The heuristic starts with an arbitrary point p and if p is determined to be a core point, the algorithm yields a cluster, which at least will contain this point p and its ϵ-neighborhood. The cluster is then expanded to include other neighboring points (core or border) which are also *density-connected* to p. When all density-connected points have been identified, the procedure jumps to the next unvisited point until visiting the whole dataset. Points that after completing this algorithm do not belong to any cluster are set to noise.

2.2 DBSCAN-like Event Detection in Twitter

Most of the DBSCAN-like techniques that have been proposed for event detection in Twitter [11, 12, 14, 15] are formulated on the basis of the Generalized DBCAN [20] algorithm, called GDBSCAN. This generalized version of DBSCAN

enables to cluster any type of spatially extended object, such as geo-located tweets.

Our approach, here, considers tweets as spatio-temporal points with a user attribute component associated to them. Following [20], we generalize the ϵ-neighborhood for two tweets t and t', $NPred(t, t')$, and the core point condition for a tweet t, $MinWeight(t)$, through two different predicates.

The predicate for the ϵ-neighborhood of a tweet t w.r.t another tweet t' combines the space-time features through two distinct ϵ_1 ϵ_2 spatio-temporal parameters,

$$NPred(t, t') \equiv dist(t_1, t'_1) \leq \epsilon_1,\ dist(t_2, t'_2) \leq \epsilon_2 \tag{1}$$

where t_1 and t_2 correspond to the spatial and temporal features, respectively. The expression $dist(t_i, t'_i)$ refers to the distances between tweet features, which we propose to be the haversine distance for the spatial dimension and the euclidean, in the time axis.

As for the core point predicate, we impose two conditions. First, the number of neighboring tweets (ϵ_1 ϵ_2-neighborhood) has to be at least $MinPts$, like in DBSCAN. Additionally, users associated to the neighboring tweets must be diverse by at least a μ percentage. These two conditions are expressed as follows,

$$MinWeight(t) \equiv |NPred(t, t')| \geq MinPts,\ UDiv(NPred(t, t')) \geq \mu \tag{2}$$

where $|NPred(t, t')|$ is the cardinality of the predicate in Eq. (1), $UDiv()$ is the percentage of unique users with respect to $MinPts$. This means that a group of tweets will be considered event if it contains at least $MinPts$ tweets and their users are at least unique in a fraction μ with respect to $MinPts$.

These two predicates correspond to those used for Tweet-SCAN [15], except that we here omitted the textual component. Nonetheless,the proposed density-aware scheme could apply to any DBSCAN-like algorithm for event detection that at least considers space-time features.

2.3 MapReduce DBSCAN

As we argued in the Introduction, performing event detection in social networks like Twitter requires to paralellize and distribute existing techniques to scale with current data volumes. Because of this, we propose to scale DBSCAN-like algorithms in a shared-nothing environment through a MapReduce approach [21].

A MapReduce algorithm for DBSCAN, named MR-DBSCAN, was proposed in [22]. This algorithm parallelizes all the critical sub-procedures of DBSCAN, which has been presented in Section 2.1. The MR-DBSCAN workflow, shown in Fig. 2, first partitions the full dataset, then performs local DBSCAN clustering in each partition, and finally merges the local clusters into global ones, which corresponds to events in our case.

An implementation of MR-DBSCAN in Apache Spark was proposed in [23] and was named RDD-DBSCAN. The main difference both algorithms is that

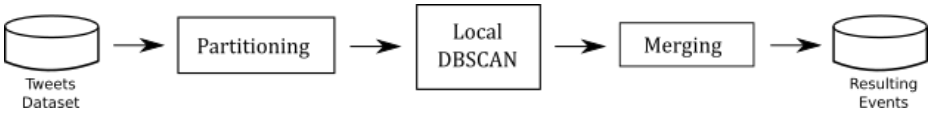


Fig. 2: Simplified MR-DBSCAN workflow

RDD-DBSCAN takes advantage of Resilient Distributed Datasets which brings data into memory to speed up computation.

Next, we review each of the MR-DBSCAN stages and highlight the main differences with respect to RDD-DBSCAN.

Partitioning MR-DBSCAN incorporates a Binary Space Partitioning (BSP) procedure to split data and distribute computation as evenly as possible. Moreover, this approach takes into account the cost of accessing disk when searching for neighboring points within the partition. On the contrary, RDD-DBSCAN simply considers the number of points per partition as the cost function, since points are already loaded into memory and the cost of accessing disk can be ignored.

The BSP partitioning in MR-DBSCAN is performed recursively in each dimension until reaching a maximum cost per partition, $maxCost$, or a minimum partition size, $MinSize$. The former condition enables to balance load among partitions while the latter is set to 2ϵ to ensure the proper functioning of DBSCAN algorithm. Given this latter restriction, the scheme divides the whole region into non-overlapped cells with side length 2ϵ. This enables to reduce the search of candidates splits among all vertical and horizontal lines aligned to cell boundaries. RDD-DBSCAN follows the same approach but it uses instead a maximum number of points per partition, $maxPts$. This variable must be set, at most, to the maximum number of points that can fit into the memory of the machine with the smallest memory available.

The partitioned regions are then enlarged by ϵ in each dimension. In this way, each partition can independently determine its core points by considering points in its ϵ-outer margin. Moreover, the overlap between partitions eases the merging stage to find proper DBSCAN clusters.

Local DBSCAN A MapReduce job performs this task. First, the mapper emits a partition ID for each point in the dataset based on the partitioning results. Second, the reducer computes the local DBSCAN for each partition as described in Section 2.1. RDD-DBSCAN performs this stage entirely in memory given that the partitioning phase has split data so that it fits in memory.

Merging This is a two-step process in which cluster merging is first computed in parallel for pairs of overlapping partitions and point types are later relabeled accordingly.

Pairs of overlapped partitions are processed parallelly to identify points that are core in both partitions or core and border respectively in each partition. These are the points whose clusters are merged and global identifiers are generated for them.

Relabeling replaces local cluster IDs into global ones and it assigns final point types (core, border, noise) to each point. Since points within the margins might be associated to different types, the relabeling strategy is to keep the more restrictive type, being the list of restrictions Core > Border > Noise.

3 Density-aware MapReduce Scheme

In the following section, we describe a novel density-aware MapReduce scheme to detect events from tweets. Although the scheme follows the MR-DBSCAN workflow from Fig. 2, individual stages have been modified according to the peculiarities of event detection in Twitter that imposes the DBSCAN-like algorithm introduced in Section 2.2.

3.1 Octree-based Partitioning

Partitioning tweet data as per its spatio-temporal features requires a three dimensional scheme to deal with the geo-location and timestamp metadata of tweets.

Although the cost-based Binary Space Partitioning (BSP) scheme proposed in [22] might be convenient for environments such as Hadoop, the cost of accessing disk becomes irrelevant in Apache Spark given that point search is entirely performed in memory, as we have seen for RDD-DBSCAN [23]. Moreover, both approaches rely on binary splits of data to balance the load among workers. These processes become very costly specially when increasing feature dimensionality (e.g. from 2D to 3D).

Because of this, we propose a naive but effective partitioning scheme based on Octree [24]. Similar to [22, 23], our proposal divides the whole space into cubes of side length $2\epsilon_1$ in the spatial and length $2\epsilon_2$ in temporal dimension, which determine the minimum partition size, $MinSize$. Following [23], we consider here a maximum number of points per partition $maxPts$ instead of a cost per partition, given that our scheme is implemented in Apache Spark.

The Octree-based partitioning is exemplified in Fig. 3 and works as follows. The spatio-temporal region containing all tweets is divided in eight equal-sized cubes and each sub-cube is recursively split as long as the $MinSize$ or the $maxPts$ conditions have not yet been achieved. The final leaves of the tree corresponds to the data partitions. As in [22, 23], partitions are then enlarged ϵ_1 and ϵ_2 in the space and time dimension, respectively.

With this partitioning scheme, we avoid the computation of the best possible split, but we might generate more partitions than necessary. Consequently, the subsequent stages might need to process and merge extra partitions increasing the execution time of these stages. However, we expect that the gain in the partitioning phase pays off the extra time in local DBSCAN and merging phases.

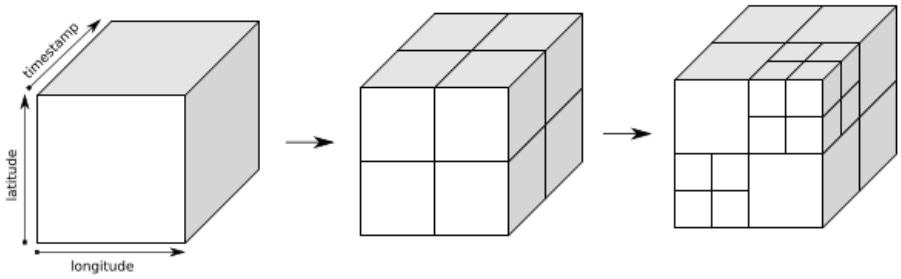


Fig. 3: Example of Octree Partitioning in spatio-temporal dimensions

3.2 Density-aware Local DBSCAN

As we mentioned earlier, one of the main limitations of DBSCAN is that fails to cluster datasets with points unevenly distributed, since DBSCAN parameters ($MinPts$, ϵ) are fixed and cannot be chosen appropriately for each sub-region.

The octree-based partitioning scheme creates spatio-temporal partitions with different density levels. These partitions correspond to different regions in space (e.g. low-density partitions are likely to be rural or deserted areas) and in time (e.g. high-density partitions are prone be at peak hours).

Moreover, the spatio-temporal properties of an event do not vary much from one place to another, or within different time periods; but the number of tweets per event certainly changes as per the user activity. Because of this, we propose to adjust the local $MinPts$ parameter based on the tweet density in each partition in such a way that the denser the partition is, the more tweets are needed to identify such event.

For a fixed set of ϵ_i parameters, we define the optimal $MinPts$ per event as the $MinPts$ value that enables to individually identify each event. We claim that there is a relationship between the optimal $MinPts$ value per event and the local density of tweets in the partition that the event belongs. As we show in Fig. 4, each event, represented by a dot, can be identified individually through an optimal $MinPts$ value that is correlated with the partition tweet density. In what follows, we explicitly assume a linear relationship (slope m, intercept b) between the local density of tweets and the $MinPts$ value.

The fitting of this linear model could be done by performing simple linear regression between the partition tweet density and the individual optimal $MinPts$. However, this approach would not necessarily optimize the overall detection accuracy. Therefore, our proposal estimates these parameters (m and b) through an optimization framework in which detection accuracy is maximized, as we will show in Section 4.4.

Introducing the density-aware parameters requires minimal changes on the local DBSCAN stage given that partition densities can be computed in the partitioning phase. Therefore, the local process simply calculates the proper

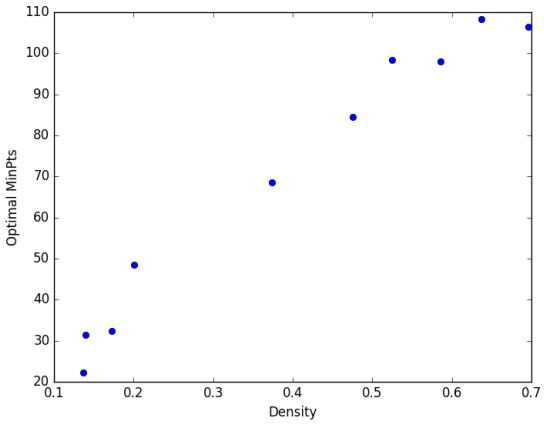


Fig. 4: Relation between the optimal $MinPts$ value per event and its partition density obtained from "La Liga" dataset in section 4.2

$MinPts$ value through the linear model (m and b) and it performs clustering with that local value through the DBSCAN-like algorithm presented in Section 2.2

3.3 Merging

The merging stage follows a two-step process as MR-DBSCAN [22] and RDD-DBSCAN [23]. Although cluster merging is performed exactly as in these algorithms, we note that the resulting clusters might be different, given that we here merge clusters from partitions with different $MinPts$ value.

On the relabeling step, we must take into account that points in the overlapping margins, which are clustered by different partitions, might be tagged with distinct point types by each partition. In contrast to MR-DBSCAN and RDD-DBSCAN, we must consider here that partitions might have different density levels. Our criteria is that the partition with higher or equal tweet density than the partition that contains the tweet will determine the point type according to the order of significance (Core > Border > Noise).

Relabeling details are shown in Algorithm 1. This algorithm takes as input a tweet or point p together with the density of its partition, and a list of labelings made by each overlapping partitions about this tweet. Note that a tweet belongs to a single partition with density $DensPartBelongs$, but it might be labeled by several partitions. For each tweet, the algorithm iterates over all possible labelings, $Values$. For a given labeling we know the cluster to which the tweet was clustered, $item.p.CluterID$, the density of its partition, $item.DensityPart$, and the flag that the tweet was associated with it, $item.p.flag$. The algorithm then checks whether the density of the partition in the given labeling is greater or equal to the density of the point's partition. If that is the case, relabeling is

Algorithm 1: Relabel Points

```
Input  : Key: (p,DensPartBelongs), Values: list of
         (p.ClusterID,DensityPart,p.flag)
Output: Key: p, Value:(ClusterID,flag)
1  ClusterID ← NULL;
2  Flag ← Noise;
3  foreach item ∈ Values do
4      if value.DensityPart ≥ DensPartBelongs then
5          if item.p.flag == Border then
6              ClusterID ← item.p.ClusterID;
7              flag ← Border;
8          else if item.p.flag == Core then
9              ClusterID ← item.p.ClusterID;
10             flag ← Core;
11             break;
12 return (p,(ClusterID,flag))
```

done as in MR-DBSCAN. Otherwise, relabeling is performed according to the partition that the tweet belongs.

4 Experimentation

4.1 Infrastructure

We conduct the experiments on a shared-nothing non-dedicated cluster with four physical machines. The cluster is managed by OpenNebula, a cloud computing platform for heterogeneous distributed data center infrastructures, through which we configured four Ubuntu virtual machines with 4 cores, 6 GB of main memory and 60 GB of hard disk space per machine. Machines are connected through a Gigabit Ethernet and Apache Spark Standalone is installed on top of them.

4.2 Datasets

For assessing the performance of the density-aware MapReduce scheme we have chosen two types of datasets: synthetic and real.

Synthetic Datasets Synthetic datasets are generated in order to test the proposed partitioning scheme under different workloads. In particular, we are interested in measuring the execution time for the Octree-based partitioning and compare against RDD-DBSCAN when increasing the dataset size.

These datasets are created using the Scikit-learn's tool samples generator utility [25]. A Python script generates datasets of points distributed according

to a Gaussian mixture model in a three dimensional euclidean space. The script requires as input the number of clusters, the number of observations and the standard deviation of each clusters to the centroid.

Since the experiment will consist in measuring execution time for different sizes of synthetic data, we created five datasets increasing in 100.000 the number of points from one to another, with a starting number of 300.000. Each synthetic data was created with twelve clusters and 1.6 of standard deviation each one.

"La Liga" Dataset A real Twitter dataset was assembled in order to validate the proposed scheme for the task of event detection. In particular, we have established a long standing connection to the Twitter Streaming API which filtered all geo-located tweets within the bounding box of the Iberian peninsula. The long standing connection was set during four days of the Spanish football league ("La Liga") April 20^{th}, 23^{th}, 30^{th} and May 8^{th} of 2016.

The aforementioned scenario provides a suitable testbed to measure the detection performance of the proposed event detection scheme. For each day, we have considered as event-related tweets all observations located in the nearby stadium area during the course of the match plus a safety period of 15 min before and after the match. Note that the space-time features for all football games will be very similar, but the number of tweets per event will directly depend on the attendance and the surrounding tweet density. In addition, we have excluded those matches that had less than 5 tweets per event and those that were outliers in terms of the relation of tweet density within the partition versus points per event.

Taking this into account, we ended with 15 events and 91.447 geo-located tweets, which we further split between training and testing following an approximated ratio of 70%/30%. In particular, we considered as training events, those ranging from April 20^{th} to 30^{th}, and testing events, those from May 8^{th} as shown in Table 1. This split also resulted in 65.231 tweets in training and 26.216 tweets in testing.

4.3 Event Detection Metrics

To evaluate the detection accuracy of our proposal, we use extrinsic clustering metrics [26]. Among all extrinsic measures, F-measure is a popular metric for this task, given that mitigates drawbacks from Purity and Inverse Purity.

The F-measure is defined as follows per each event E_i and cluster C_j,

$$F(E_i, C_j) = 2 \cdot \frac{Recall(E_i, C_j) \cdot Precision(E_i, C_j)}{Recall(E_i, C_j) + Precision(E_i, C_j)} \tag{3}$$

where precision is the proportion of tweets from cluster C_j that are tagged as event E_i. Oppositely, recall is the proportion of tweet from event E_i that are clustered as C_j. The following expressions formally define precision and recall per each pair of event and cluster.

$$Precision(C_j, E_i) = \frac{\mid C_j \cap E_i \mid}{\mid C_j \mid} \qquad Recall(C_j, E_i) = \frac{\mid E_i \cap C_j \mid}{\mid E_i \mid} \tag{4}$$

Table 1: Training and testing football events

Event	Stadium	Event date	Start time	Points
1	Riazor	20 April	19:45	17
2	San Mames	20 April	20:30	16
3	Bernabeu	20 April	21:45	66
4	Mestalla	20 April	20:30	15
5	Rosaleda	20 April	20:30	6
6	Calderon	23 April	18:00	16
7	Camp Nou	23 April	20:15	69
8	Calderon	30 April	18:00	27
9	Benito Villamarín	30 April	20:15	16
10	Anoeta	30 April	15:45	7
11	Los Cármenes	30 April	21:50	7
12	Camp Nou	8 May	16:45	93
13	Ciudad de Valencia	8 May	16:45	9
14	Sánchez Pizjuan	8 May	16:45	24
15	Balaidos	8 May	16:45	15

Finally, the F-measures from Eq. (3) are combined through a weighted average scheme across all events. For each event, the maximum F-measure with respect to all clusters is considered for averaging.

$$F = \sum_i \frac{\mid E_i \mid}{N} max_j \, F(E_i, C_j) \tag{5}$$

where N is the total number of tweets.

Purity and Inverse Purity are defined similarly as the weighted average across clusters and events, respectively. While Purity considers the maximum precision w.r.t events, Inverse Purity uses the maximum recall w.r.t. clusters. However, both figures by themselves fail to measure proper clustering. On the one hand, Purity penalizes the noise in a cluster, but it does not reward grouping tweets from the same event together. For example, if we simply make one cluster per observation, we reach trivially a maximum purity value. On the other hand, Inverse Purity rewards grouping tweets together, but it does not penalize mixing items from different events. Here, we can reach maximum Inverse Purity by making a single cluster with all tweets.

Therefore, we consider the F-measure from Eq. (5) for assessing the event detection performance.

4.4 Evaluation

Execution Times To validate the goodness of the proposed Octree-based partitioning scheme, we compare its execution time against BSP-based partitioning in different-sized synthetic datasets.

For both schemes, we set algorithm parameters so that all clusters could be discovered at every experiment. Therefore, $\epsilon 1$, $\epsilon 2$ was set to 0.01, $maxPoints$ to 5000 and $MinPts$ to 75. For each experiment, we collected the partitioning time, the clustering and merging time and the total execution time.

Results, Table 2 and Fig. 5, show that the total execution time of the Octree-based approach overcomes the BSP-based approach proposed in RDD-DBSCAN [23]. This improvement is clearly achieved in the partitioning phase due to the fact that Octree partitioning is less expensive than BSP partitioning in computation terms. As expected, the gain in the partitioning phase comes at the expense of an increase at the clustering and merging phase.

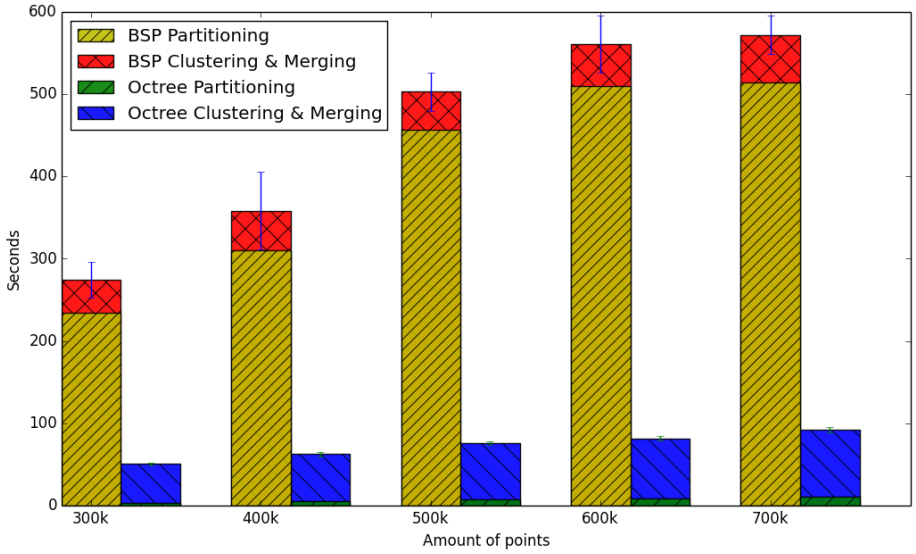


Fig. 5: Execution times for BSP-based and Octree-based MapReduce DBSCAN

Table 2: Execution times for BSP-based and Octree-based MapReduce DBSCAN

Phase/Data	300k	400k	500k	600k	700k
BSP Partitioning	234.28	310.17	456.55	509.46	513.75
Octree Partitioning	3.17	5.16	7.13	8.40	10.55
BSP Clustering and Merging	39.97	48.08	46.45	51.04	58.00
Octree Clustering and Merging	47.23	57.14	68.87	73.10	81.35
BSP total execution	274.25	358.25	503.00	560.50	571.75
Octree total execution	50.40	62.30	76.00	81.50	91.90

Detection Performance To validate the detection accuracy of the proposed density-aware scheme, we compare the best performing DBSCAN-like model against the density-aware scheme in "La Liga" dataset.

Given that all labeled events shared similar space-time properties (tweets were located nearby stadiums and during the course of football matches), ϵ_1 and ϵ_2 parameters were out of the optimization scope and they were assumed known and constant for all experiments. In particular, we found out that an $\epsilon_1 = 500m$ and $\epsilon_2 = 1hour$ performed reasonably well in this dataset.

Therefore, we aimed to find the best performing configurations of both models by optimizing the $MinPts$ parameter for the DBSCAN-like model and the linear parameters (m, b) for the density-aware scheme. The optimizations were performed in "La Liga" training dataset, while the test dataset was used to validate the values found. A greater F-measure value of the density-aware scheme in the training and testing datasets would indicate that our proposal outperforms the basic DBSCAN-like approach.

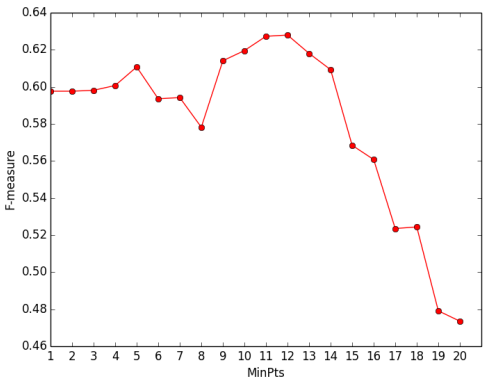


Fig. 6: $MinPts$ optimization for DBSCAN-like algorithm

Fig. 6 shows the optimization of the $MinPts$ parameter for a DBSCAN-like algorithm in terms of F-measure in the training set. As it is depicted, the global maximum is achieved for a $MinPts$ of 12 with a overall F-measure value of 0.628.

Fig. 7 shows the optimization of the linear parameters (m, b) in terms of F-measure for the density-aware scheme. The left figure plots the optimization results in a suitable view to understand the variation of the slope m. The right figure plots the same optimization results in a different angle of view to understand the intercept b. In both axis, a global optimum seems to exist given that extremely high or low slope and intercept values will end up with lower F-measure values. Therefore, the best performing linear model for the density-

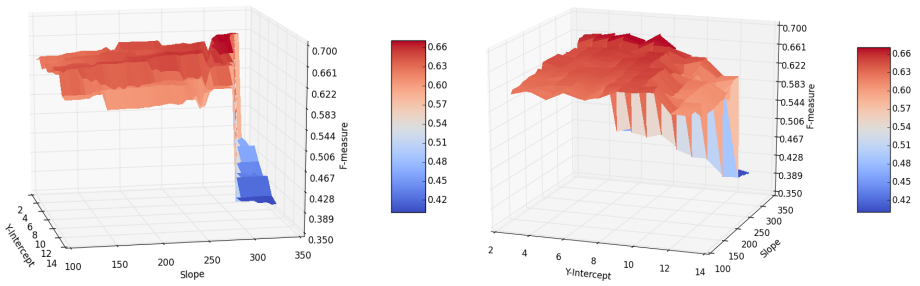
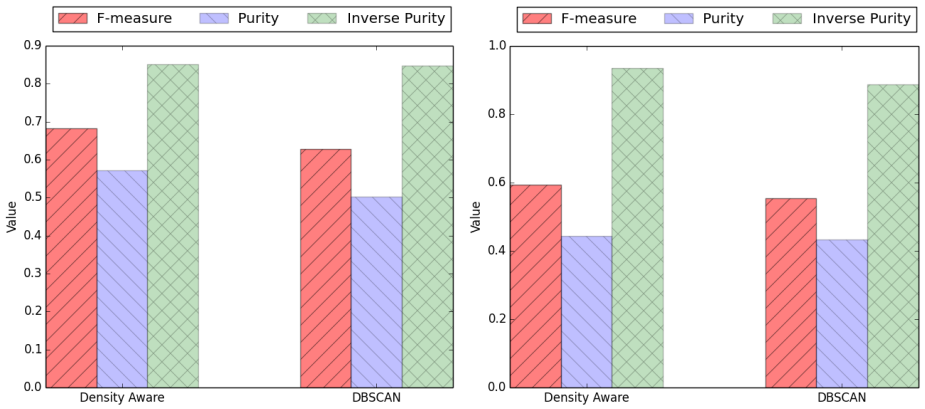


Fig. 7: Linear (m, b) optimization for density-aware scheme

aware scheme consist of a slope $m = 300$ and a intercept $b = 8$, which results in a F-measure of 0.683.

With this optimal values we now compare the performance of both algorithms in the testing dataset. Results shown in Fig. 8 sustain that the detection performance obtained in the density-aware scheme are higher than the ones of the the basic DBSCAN-like algorithm in both training and testing datasets. In both cases, the percentage of improvement is around 5%, being a promising value for future work in the field. In addition, the global purity and inverse purity values computed for both algorithms and datasets reveal that our proposal is increasing both measures with respect to the traditional algorithm, not prioritising one over the other.



(a) Training dataset (b) Testing dataset

Fig. 8: F-measure, Purity and Inverse Purity values.

5 Conclusions

In this paper, we identified two major shortcomings when scaling DBSCAN-like algorithms for event detection systems in Twitter. First, the large amount of tweets hampers the use of event detection techniques that run into a single machine. Second, the geographical or temporal scaling of these systems has to explicitly consider that tweeting activity varies in space and time.

To tackle both shortcomings, we proposed a density-aware MapReduce scheme which benefits from local DBSCAN computations to tune its local parameters to the neighboring tweet densities. The rationale for using density-aware parameters is that events in highly dense regions are likely to contain more tweets, while those in low-density regions will contain less.

The assessment of the proposed scheme is performed in a dataset of geolocated tweets in the Iberian peninsula during the course of several football matches. Tweets nearby the stadium during the football game are manually identified as event-related tweets. The evaluation shows that our proposal to incorporate density awareness outperforms classical DBSCAN techniques. Moreover, we also show that the overall execution time improves with respect to RDD-DBSCAN by using a naive but effective Octree-based partitioning scheme.

6 Future Work

The proposed density-aware MapReduce scheme for event detection has been evaluated in a dataset of events which all shared similar spatio-temporal features. Future work should address the evaluation in datasets of heterogeneous events. For example, events that last many hours but are located within a narrow area or events that last few hours but are geographically very spread.

Similarly, we have observed that some clusters detected by our event detection approach did not correspond to real-world events, but to popular places known as landmarks, such as the city centre. In order to avoid detecting these clusters, a textual component could be added in the neighborhood search for selecting tweets which are similar in meaning, as in [14, 16]. However, searching for textual objects will cause extra computational cost that might not be disregarded by future research.

As in RDD-DBSCAN, the need to load the complete data set for a given partition into memory still remains open. Future work should focus on this given that the use of disk and memory will bring new ideas to scale up event detection in Apache Spark. For example, we might need to rethink the proposed Octree-based partitioning scheme so that it takes into account the extra-cost to now load data from disk.

References

1. W. Wong, and D. Neill: Tutorial on Event Detection. Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) (2009).

2. M. Kulldorff, W. Athas, E. Feurer, B. Miller and C. Key: American journal of public health 9, 1377–1380 (1998)
3. Z. Yu : Tutorial on Location-Based Social Networks. Proceedings of the 21st international conference on World wide web (WWW) (2012).
4. T. Sakaki, M. Okazaki and Y, Matsuo: Earthquake shakes Twitter users: real-time event detection by social sensors. Proceedings of the 19th international conference on World Wide Web (WWW) (2010).
5. R. Lee and K. Sumiya: Measuring Geographical Regularities of Crowd Behaviors for Twitter-based Geo-social Event Detection. Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks (LBSN) (2010).
6. N. Newman: Mainstream media and the distribution of news in the age of social discovery. Reuters Institute for the Study of Journalism, University of Oxford (2011).
7. B. Stelter and N. Cohen: Citizen Journalists Provided Glimpses of Mumbai Attacks. `http://www.nytimes.com/2008/11/30/world/asia/30twitter.html` (2008)
8. F. Atefeh and W. Khreich: A survey of techniques for event detection in twitter. Computational Intelligence 1, 132–164 (2015).
9. H. Becker, M. Naaman and L. Gravano: Beyond Trending Topics: Real-World Event Identification on Twitter. Proceedings of the Fifth International Conference on Weblogs and Social Media (2011)
10. M. Ester, H. Kriegel, J. Sander and X. Xu: A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd. Vol. 96. No. 34. (1996).
11. J. Gomide, A. Veloso, W. Meira, V. Almeida, F. Benevenuto, F. Ferraz and M. Teixeira: Dengue Surveillance Based on a Computational Model of Spatio-temporal Locality of Twitter. Proceedings of the 3rd International Web Science Conference.
12. K. Tamura and T. Ichimura: Density-based spatiotemporal clustering algorithm for extracting bursty areas from georeferenced documents. IEEE International Conference on Systems, Man, and Cybernetics (SMC) (2013).
13. D. Birant and A. Kut: ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data and Knowledge Engineering (2007).
14. S. Singh: Spatial Temporal Analysis of Social Media Data. Master Thesis at Technische Universität München (2015).
15. J. Capdevila, J. Cerquides, J. Nin and J. Torres. Tweet-SCAN: An event discovery technique for geo-located tweets. Artificial Intelligence Research and Development - Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence (2015).
16. J. Capdevila, J. Cerquides, J. Nin and J. Torres. Tweet-SCAN: An event discovery technique for geo-located tweets. Pattern Recognition Letters. Available online 25 August (2016).
17. D. Blei: Probabilistic topic models. Communications of the ACM. Vol. 55 No. 4 77–84 (2012).
18. L. Li, M. Goodchild and B. Xu: Spatial, temporal, and socioeconomic patterns in the use of Twitter and Flickr. Cartography and Geographic Information Science. Vol. 40 No. 261–77 (2013).
19. M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker and I. Stoica: Spark: cluster computing with working sets. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (2010).
20. J. Sander, M. Ester, H Kriegel and X. Xu: Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. Data Mining and Knowledge Discovery. Vol. 2 No. 2 169-194 (1998).
21. J. Dean and S. Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. Communications ACM Vol. 51 107–113 (2008).

22. Y. He, H. Tan, W. Luo, S. Feng and J. Fan: MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data. Frontiers of Computer Science. Vol. 8 83–99 (2014).
23. I. Cordova and T. S. Moh: DBSCAN on Resilient Distributed Datasets. International Conference on High Performance Computing Simulation (HPCS) 531-540 (2015).
24. D. Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.
25. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. Vol. 12 2825–2830 (2011).
26. E. Amigó, J. Gonzalo, J. Artiles and F. Verdejo: A comparison of extrinsic clustering evaluation metrics based on formal constraints. Information Retrieval Vol. 12 No. 4 461-486 (2009).