BigCAB: Distributed Hot Spot Analysis over Big Spatio-temporal Data using Apache Spark (GIS Cup)

Panagiotis Nikitopoulos
Department of Digital Systems
University of Piraeus
18534, Piraeus, Greece
nikp@unipi.gr

Aris-Iakovos Paraskevopoulos
Department of Digital Systems
University of Piraeus
18534, Piraeus, Greece
arisparaskevop@gmail.com

Christos Doulkeridis
Department of Digital Systems
University of Piraeus
18534, Piraeus, Greece
cdoulk@unipi.gr

Nikos Pelekis
Department of Statistics and Ins. Science
University of Piraeus
18534, Piraeus, Greece
npelekis@unipi.gr

Yannis Theodoridis
Department of Informatics
University of Piraeus
18534, Piraeus, Greece
ytheod@unipi.gr

ABSTRACT

Hot spot analysis is the problem of identifying statistically significant spatial clusters from an underlying data set. In this paper, we target the problem of hot spot analysis of massive spatio-temporal data, which raises the need for a parallel and scalable solution that operates on data distributed over a set of nodes. We propose an algorithm, called *BigCAB*, implemented in Spark, that solves the problem in a parallel and scalable way. Our experiments on real data representing taxi trips demonstrate both the efficiency as well as the nice scaling properties of our algorithm.

CCS Concepts

•Computing methodologies → MapReduce algorithms; •Information systems → Geographic information systems; Parallel and distributed DBMSs;

Keywords

Hot spot analysis; parallel and distributed processing; spatio-temporal data; MapReduce; Apache Spark

1. INTRODUCTION

Massive amounts of spatio-temporal data are produced on a daily basis, including mobile objects' trajectories, tweets by mobile Twitter users, check-ins in Foursquare, etc. Analyzing this wealth of spatio-temporal data has the potential to discover hidden patterns or result in non-trivial insights. In this context, a useful data analysis task is *Hot Spot Analysis*, which is the process of identifying statistically significant clusters. The problem is well-studied in spatial databases and medium-sized data sets. However, to the best of our knowledge, there is a lack of parallel processing solutions that operate on distributed spatio-temporal data in an efficient and scalable way.

Motivated by this shortcoming, in this paper, we study the problem of identifying statistically significant clusters over massive spatio-temporal data, which is stored in a distributed way. To address the GIS Cup 2016 Challenge related with the immense volume of underlying data, we design and develop a scalable solution in a parallel data processing framework, namely Apache Spark, which is currently the most widely established framework for batch processing and analytics. As such, the main contribution of our work is the design and implementation of one of the first parallel algorithms that solves the problem of hot spot analysis in a scalable way.

The remainder of this paper is structured as follows: Section 2 formulates the problem under study and provides details on the GIS Cup 2016 Challenge. Section 3 presents the proposed algorithm and its implementation details. Then, in Section 4, we present some experimental results and in Section 5 we briefly describe recent works in hot spot analysis. Finally, Section 6 concludes the paper.

2. PROBLEM FORMULATION

Consider a spatio-temporal data set $\mathcal{D}$ that consists of data points $p \in \mathcal{D}$ described by 2D spatial coordinates ($p.x$ and $p.y$), a timestamp ($p.t$), as well as any other information related to the spatio-temporal position of p. A data point p also contains an *attribute value* ($p.v$) that is application-specific and denotes a numerical quantity associated with each data point, which is going to be used for data analysis purposes. Furthermore, consider a spatio-temporal partitioning $\mathcal{P}$ which partitions the 3D spatio-temporal domain into n 3D cells $\{c_1, \ldots, c_n\} \in \mathcal{P}$. There is a one-to-one mapping between a data point p and a cell c_i, which is determined based on p being enclosed in c_i. We also define the attribute value x_i of the i-th cell as: $x_i = \sum_{p \in c_i} p.v$. Table 1 provides an overview of the notation.

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA

DOI: http://dx.doi.org/10.1145/2996913.3004065

Symbol	Description
$\mathcal{D}$	Spatio-temporal data set
$p \in \mathcal{D}$	Spatio-temporal data point $(p.x, p.y, p.t)$
$p.v$	Attribute value of data point p
$\mathcal{P}$	3D space partitioning $\mathcal{P} = \{c_1, \ldots, c_n\}$
c_i	The i-th cell of partitioning $\mathcal{P}$, $(1 \leq i \leq n)$
x_i	Attribute value of cell $c_i \in \mathcal{P}$
n	The number of cells in $\mathcal{P}$
G_i^*	The Getis-Ord statistic for cell c_i
$\mathcal{N}(c_i)$	The direct neighboring cells of cell c_i
top-k	Requested number of most significant cells

Table 1: Overview of symbols.

The problem of *hot spot analysis* addressed in this work is to identify statistically significant spatio-temporal areas (i.e., cells of $\mathcal{P}$), where the significance of a cell c_i is a function of the cell's attribute value x_i, but also of other neighboring cells' attribute values. A commonly used function (statistic) is the Getis-Ord statistic G_i^*, defined as [5]:

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j} x_j - \overline{X} \sum_{j=1}^{n} w_{i,j}}{S \sqrt{\frac{[n \sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2]}{n-1}}} \quad (1)$$

where x_j is the attribute value for cell j, $w_{i,j}$ is the spatial weight between cell i and j, n is equal to the total number of cells, and:

$$\overline{X} = \frac{\sum_{j=1}^{n} x_j}{n} \quad (2)$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\overline{X})^2} \quad (3)$$

Based on this, the problem of hot spot analysis is to identify the k most statistically significant cells according to the Getis-Ord statistic, and can be formally stated as follows.

PROBLEM 1. *(Hot spot analysis) Given a spatio-temporal data set $\mathcal{D}$ and a space partitioning $\mathcal{P}$, find the top-k cells $TOPK = \{c_1, \ldots, c_k\} \in \mathcal{P}$ based on the Getis-Ord statistic G_i^*, such that: $G_i^* \geq G_j^*$, $\forall c_i \in TOPK$, $c_j \in \mathcal{P} - TOPK$.*

2.1 The GIS Cup 2016 Challenge

In this paper, we study an instance of Problem 1 based on the GIS Cup 2016 Challenge[1], where the aim is to perform hot spot analysis over massive spatio-temporal data by proposing a parallel and scalable solution. Thus, we turn our attention to large-scale data sets that exceed the storage and processing capabilities of a single centralized node. Therefore, we assume that the data set $\mathcal{D}$ is stored distributed in multiple nodes, horizontally partitioned, without any more specific assumptions about the exact partitioning mechanism. Put differently, a node stores a subset D_i of the records of $\mathcal{D}$, and it holds that $D_i \bigcap D_j = \emptyset$ (for $i \neq j$), and $\bigcup D_i = \mathcal{D}$. Hence, in this paper, we study a distributed version of Problem 1.

Moreover, some simplifying assumptions were made, as indicated by the Challenge description. First, for the computation of the Getis-Ord statistic of a cell c_i, only the direct neighboring cells are considered, instead of all n cells of $\mathcal{P}$. We use $\mathcal{N}(c_i)$ to denote this set of neighboring cells of c_i, and $|\mathcal{N}(c_i)|$ denotes the number of such neighboring cells[2]. Second, to simplify the computation of Getis-Ord statistic, the weight $w_{i,j}$ of each neighbor cell c_j to cell c_i is presumed to be equal to one. Hence, we obtain the following simplified formula for the Getis-Ord statistic:

$$G_i^* = \frac{\sum_{j \in \mathcal{N}(c_i)} x_j - \overline{X} \cdot |\mathcal{N}(c_i)|}{S \cdot \sqrt{\frac{n \cdot |\mathcal{N}(c_i)| - |\mathcal{N}(c_i)|^2}{n-1}}} \quad (4)$$

3. BIGCAB ALGORITHM

In this chapter, we present our solution for distributed hot spot analysis over big spatio-temporal data. The proposed algorithm, called BigCAB, is designed to be efficiently executed over a set of nodes in parallel and utilizes the MapReduce framework of Apache Spark. The input data set $\mathcal{D}$ is stored in a distributed file system, in particular HDFS.

3.1 Overview

Intuitively, our solution consists of three main steps. As the input data set $\mathcal{D}$ is stored distributed in different nodes, the first step is to re-partition this data based on the cell $c_i \in \mathcal{P}$ where each data point belongs to. Thus, for any accessed data point p, we project it to the cell c_i that encloses p. We discard any other information regarding p, apart from the attribute value $p.v$. This grouping of data points per cell, allows us to compute the attribute value of each cell, i.e., $x_i = \sum_{\forall p \in c_i} p.v$. Subsequently, in the second step, the attribute value x_i of each cell c_i is aggregated with the attribute values of the neighboring cells $\mathcal{N}(c_i)$. In the last step, we compute the statistic of each cell c_i, by utilizing the simplified Getis-Ord formula of Equation 4.

The above description explains the rationale of our approach. In the following, we describe the implementation of our solution (BigCAB) in Spark, along with the necessary technical details.

3.2 Spark Implementation

Apache Spark is a popular framework that utilizes MapReduce for efficiently executing parallel tasks. It is considered as one of the best solutions for batch processing tasks, largely overcoming some of the limitations of Hadoop [1]. Hence, it is appropriate for implementing and executing the BigCAB algorithm. Spark uses the abstraction of *resilient distributed data set (RDD)* [6], which is a distributed collection of elements. Any work that has to be carried out in Spark is expressed as RDD creation, RDD transformation, or operation on RDDs.

Our implementation of BigCAB in Spark is depicted in Algorithm 1. It consists of two MapReduce phases followed by a top-k operation. Between the two MapReduce phases, a *for each* clause is used to compute some statistical information. Note that in the provided pseudocode, we use lambda expressions syntax to eliminate verbosity.

In lines 2–4, the first *flatMapToPair* function reads the input files from HDFS, and transforms data points p into key-value pairs. Keys are simple integer variables, generated according to p's spatio-temporal projection on the cells

[1] http://sigspatial2016.sigspatial.org/giscup2016/

[2] In the 3D space, a cell c_i typically has $|\mathcal{N}(c_i)| = 26$ neighboring cells, unless the cell is located at the space boundaries.

Algorithm 1 BigCAB algorithm

```
1: procedure BigCAB
2: gridRDD = filesOnHDFS.flatMapToPair(p =>
3:     emit new pair(getCellId(p), p.v)
4: ).reduceByKey(p1, p2 => emit p1.v + p2.v)
5: gridRDD.forEach(ci => update accumulators)
6: neighborRDD = gridRDD.flatMapToPair(ci =>
7:     N(ci) = getDirectNeighborIds(ci)
8:     for each j in N(ci) do
9:         emit new pair(j, xi)
10:    end for
11: ).reduceByKey(x1, x2 => emit x1 + x2)
12: scoresRDD=neighborRDD.mapPartitionsToPair(ci=>
13:    for each local ci in neighborRDD do
14:        list.add(new pair(ci, G*i))
15:        sort list and keep only the top-k pairs
16:    end for
17:    emit list)
18: sort scoresRDD and return the top-k cells
19: end procedure
```

of $\mathcal{P}$; i.e., for a data point $p \in c_i$ the key is the identifier i of the corresponding cell c_i. Values represent the respective attribute value $p.v$ of p. Notice that by using primitive key types we improve BigCAB's efficiency and decrease communication overhead between the nodes in the cluster. When all data are loaded and transformed into key-value pairs, they are aggregated by a *reduceByKey* function, whose task is to compute the sum of attribute values x_i for each cell c_i.

Then, in line 5, a *forEach* function is used to compute the statistical values of Equations 2 and 3 over the cells of $\mathcal{P}$. To this end, we create two accumulators that will provide the sum ($\sum_{i=1}^{n} x_i$) and squared sum ($\sum_{i=1}^{n} x_i^2$) of the attribute values x_i of cells c_i respectively. Notice that according to the documentation of Apache Spark, accumulators should not be modified by a *map* function, as this may lead to unexpected behavior.

The next step (lines 6–11) is to compute the neighbor aggregated attribute values ($\sum_{j \in \mathcal{N}(c_i)} x_j$), as mentioned in the previous section. This process is performed by the second MapReduce phase. We create $|\mathcal{N}(c_i)|$ number of copies for each cell c_i of $\mathcal{P}$, where each copy's key is altered to match the key of a direct neighboring cell. These copies are aggregated by a *reduceByKey* function, where the sum of the attribute values of each cell's direct neighbors is computed.

In order to compute the G_i^* scores, we apply the simplified formula of Equation 4 to each point produced by the previous step (lines 12–17). This task is accomplished by a *mapPartitionsToPair* function which processes the local cells of each processing node. Since cells are assigned to nodes based on the available hardware resources, it suffices for each node to report only its local top-k cells based on G_i^* score. Thus, this function also performs a top-k query on the locally computed cells. As such, each node emits to the coordinator node only its k most statistically significant cells by score. Then, the coordinator produces the final result by reporting only the k cells with highest scores (line 18).

4. EXPERIMENTAL EVALUATION

In this section we evaluate the performance of BigCAB, which is implemented in Java, using Apache Spark.

Parameter	Values
Input size (GB)	6, 12 **24**
No. of cells in $\mathcal{P}$ (n)	12K, 100M, **800M**

Table 2: Experimental parameters and values.

4.1 Experimental Setup

Platform. We deployed our algorithm in an in-house CDH cluster consisting of sixteen (16) server nodes. Each of the nodes d1-d8 has 32GB of RAM, 2 disks for HDFS (5TB in total) and 2 CPUs with a total of 8 cores running at 2.6 GHz. The nodes d9-d12 have 128GB of RAM, 4 disks for HDFS (8TB in total) and 2 CPUs with a total of 12 cores (24 hyperthreads) running at 2.6 GHz. Finally, each of the nodes d13-d16 is equipped 128GB RAM, 4 disks for HDFS (8TB in total), and 2 CPUs with a total of 16 cores running at 2.6GHz. Each of the servers in the cluster function as DataNode and NodeManager, while one of them in addition functions as NameNode and ResourceManager. Each node runs Ubuntu 12.04. We use the CDH 5.4.8.1 version of Cloudera and Oracle Java 1.7. We configured HDFS with 128MB block size and used a default replication factor of 3.

Data Set. Our algorithm was tested against a real data set. By following the guidelines of the GIS Cup 2016 Challenge, we used a very large collection of spatio-temporal observational data; the New York City Taxi and Limousine Commission Yellow Cab trip data[3]. This data set consists of over one billion records representing all Yellow Cab taxi trips in New York City between January 2009 and June 2016. Each record in the data set contains key information such as pickup and dropoff date, time, and location (latitude, longitude), trip distance, passenger count, and fare amount. We use passenger count as attribute value of a record.

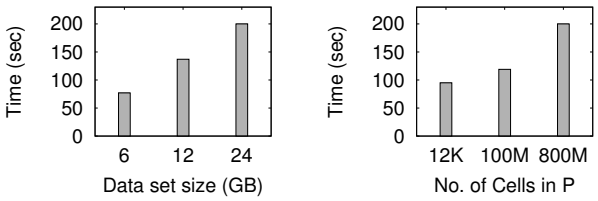
In this study, we used a subset of the available data, corresponding to the taxi trips of the year 2015. This data is split to 12 individual files (one for each month), making a total size of approximately 24GB. Furthermore, the source data were clipped to an envelope (bounding box), encompassing the five New York City boroughs, in order to remove some of the noise or erroneous values in the data set. This envelope was defined according to the following coordinates: latitude 40.5N - 40.9N, longitude 73.7W - 74.25W.

Grid Characteristics. A 3D grid ($\mathcal{P}$) was used to aggregate the spatio-temporal data to cells. The properties of the grid, i.e., cell size in each dimension, can be specified at runtime. In this study, we use a default cell size of 0.001 degrees latitude and longitude, and roughly 2.5 hours of time. This results to approximately 800 million cells ($n = 800M$) for our default setting.

Query. The goal of the GIS Cup 2016 Challenge was to identify the fifty (k=50) most statistically significant dropoff locations (cells of the grid) by passenger count in both time and space using the Getis-Ord statistic.

Evaluation Methodology. In order to demonstrate the efficiency and scalability of BigCAB, we conducted a set of experiments. We monitored the execution time needed for each experiment to complete, while varying the size of input data and the size of grid cells. The parameter values are shown in Table 2, with default values in bold.

[3]http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml



(a) Input size (b) Cells in $\mathcal{P}$

Figure 1: Performance of BigCAB.

4.2 Experimental Results

Varying data set size. We evaluated the execution time of BigCAB for different sizes of the data set to test its scalability with input size.

Figure 1(a) depicts the results of this experiment. Obviously, having to process larger input sizes results in higher execution time. More specifically, an input size of 6GB requires 77 seconds to be processed, while the 24GB input size requires 200 seconds. Put differently, when we quadruple the input size, the execution time does not increase four times, but less. These results indicate that BigCAB has some nice scaling properties.

Varying grid cell size. In the next experiment, we assessed the sensitivity of BigCAB for different cell sizes of the grid $\mathcal{P}$. By using a larger cell size, the results are aggregated into larger groups of time and space, providing the user a less granulated information.

Figure 1(b) shows that having a larger number of cells results in higher execution time, due to increased processing complexity in some stages of BigCAB. This is expected since more cells result in more key-value pairs being created in both MapReduce phases. Thus larger RDDs need to be processed, and also the score computation phase is more expensive. By requesting results of smaller granularity, BigCAB achieves better performance results. However, this may have a negative impact on the accuracy of hot spot discovery.

5. RELATED WORK

Hot spot analysis is a special case of spatio-temporal data analysis and mining, which has been the object of many research efforts lately. Recent works on hot spot analysis for spatio-temporal data include [2, 4].

The study in [4] proposes a different way to visualize and analyze spatio-temporal data. The aim is to identify areas of high event density, by using multivariate kernel density estimation. Different kernels in spatial and temporal domains can be used. After such hot spots have been identified, a spatio-temporal graph can be formed to represent topological relations between hot spots.

In [2], a spatio-temporal graph is analyzed in order to find anomalies on people's regular travel patterns. These interesting phenomena are called *black holes* and *volcanos*, represented as subgraphs with overall inflow traffic greater than the overall outflow by a threshold and vice-versa. The detection of frequent patterns and relations between black holes and volcanos lead to the discovery of human mobility patterns.

In [3], hot spot analysis is used for studying mobile traffic. The aim is to identify locations where the density of data volumes transmitted is high, based on specific values of thresholds. The results of the analysis are then used to detect the distribution of mobile data traffic hot spots and to propose a meaningful cell deployment strategy.

6. CONCLUSIONS

In this paper, we proposed a parallel and distributed solution for hot spot analysis over big spatio-temporal data. The problem is challenging due to the immense volume of input data, which makes centralized solutions inefficient or impractical. We presented the design of a parallel algorithm, called BigCAB, which is implemented in Apache Spark. Our experimental study on real data demonstrates the scalability and the efficiency of BigCAB. In our future work, we intend to apply our approach to other domains, such as the maritime or aviation domains, in order to discover hot spot areas for vessels and aircrafts respectively.

7. ACKNOWLEDGMENTS

This work has been partly supported by the University of Piraeus Research Center; also by project datACRON, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687591.

8. REFERENCES

[1] C. Doulkeridis and K. Nørvåg. A survey of large-scale analytical query processing in mapreduce. *VLDB J.*, 23(3):355–380, 2014.

[2] L. Hong, Y. Zheng, D. Yung, J. Shang, and L. Zou. Detecting urban black holes based on human mobility data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '15, pages 35:1–35:10, New York, NY, USA, 2015. ACM.

[3] H. Klessig, V. Suryaprakash, O. Blume, A. J. Fehske, and G. Fettweis. A framework enabling spatial analysis of mobile traffic hot spots. *IEEE Wireless Commun. Letters*, 3(5):537–540, 2014.

[4] J. Lukasczyk, R. Maciejewski, C. Garth, and H. Hagen. Understanding hotspots: A topological visual analytics approach. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '15, pages 36:1–36:10, New York, NY, USA, 2015. ACM.

[5] J. K. Ord and A. Getis. Local spatial autocorrelation statistics: Distributional issues and an application. *Geographical Analysis*, 27(4):286–306, October 1995.

[6] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 15–28, 2012.