Hot Spot Analysis

Harsha Illuri
ASU ID:1213218082
hilluri@asu.edu
Arizona State University, Tempe

Manjusha Ravindranath
ASU ID:1212109143
mravind1@asu.edu
Arizona State University, Tempe

Rohith Reddy
ASU ID:1212908292
rvajrala@asu.edu
Arizona State University, Tempe

ABSTRACT

In the paper Hot Spot Analysis, we perform spatial analysis-hot range and cell analysis of New York taxi cabs. Hot zone analysis performs range join operations on rectangle datasets and point datasets. For each rectangle, number of points located within the rectangle is analysed. The more the points in the rectangle hotter the rectangle. Hot cell analysis is about applying spatial statistics to identify hotspots. SparkSQL which is an integral part of Apache Spark offers a tight coupling between relational and procedural processing.

CCS CONCEPTS

• Spark → SparkSQL;

KEYWORDS

Spark, SparkSQL, RDD, Hot Zone, Hot Spot, Hot cell, Range query, Distance query.

1 INTRODUCTION

In the last decade, there has been a lot of research activity in studying geospatial data for a variety of applications from weather maps, social, economic and industrial data to identify and control epidemics [1]. It's a GPS World now, world of mobile computing. We need to know the location of physical objects in space. There are mobile apps to measure marine environments for fishery, habitat planning and conservation. The NASA program 'The Earth Observing System Data and Information System' is an Earth Science Data Systems Program to manage NASA's earth science data from various sources-satellites, flight and field measurements that has been around from the 1990s. [2] The program researches the activities on the Earth, properties on spatial and temporal scales to understand natural and manmade processes, so we are better suited to predict its evolution.

Spatial analysis presents challenges in multiple levels.

NASA EOSDIS archive data is about 3 PB and they generate 5 TB of data each day. Google produces 25 PB of data per day. This makes research necessary to analyze and organize the big data. [5] Spatial data mining algorithms are needed to find useful patterns in this data explosion. Computational complexity increases with any dependency or correlation between spatial and temporal measurements.

Applications like biomass monitoring [1] over large geographic regions by NASA's Terra satellite with the MODIS instrument contains about 3600 data points at daily temporal resolution. Image based content search pattern and retrieval tasks require feature extraction and classification tasks. We need new models that model spatial and temporal limitations efficiently that have higher quality and an order of magnitude faster than current techniques. [5]

Clusters of cases of rare diseases can be detected by spatial analysis. Clusters can be detected over a large area or small area affected by nuclear trials for example. [9] Disaster management and evacuation coordination during natural hazards are applications that can be expedited by spatial analysis. Man-made hazards like terrorism also can benefit from geographic information system analysis. [10]

All above applications and more, even the very existence of earth and human kind motivate spatial analysis.

2 ARCHITECTURE AND IMPLEMENTATION DETAILS

2.1 Spark

The current cluster computing techniques like MapReduce lack abstractions for leveraging distributed memory. This makes them inefficient for iterative machine learning and graph applications that reuse intermediate results across multiple computations. This is where Spark comes into play. In most current frameworks, the only way to reuse data between computations for example between two MapReduce jobs is to write it to an external storage system like a distributed file system. This can cause a lot of overhead with data replication and disk I/O.

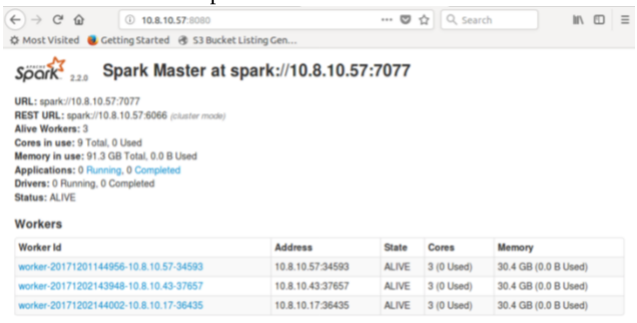

Fig 1 Spark Mater UI

The current frameworks developed to solve this problem do not provide abstractions for general reuse to let an user load several datasets into memory and run ad-hoc queries across them. Also existing in-memory abstractions such as distributed shared

memory, key value store databases offer an interface based on fine grain updates.

In Spark a new abstraction called Resilient distributed datasets(RDDs) is proposed which enables efficient data reuse in a broad range of applications. RDDs are fault tolerant parallel distributed data structures that let users explicitly persist intermediate results in memory. RDD is a logical reference of a dataset which is partitioned across many server machines in the cluster.

RDDs have the following properties:

1. Immutability and partitioning: RDDs are composed of partitions of a collection of records. Users can also control the partitioning to optimize data replacement and manipulate the RDDs using a rich set of operators.
2. Coarse Grained Operations: Operations like map, filter or group By operation which will be performed on all elements in a partition of RDD.
3. Fault Tolerance: RDDs are created over a set of transformations, it logs those transformations rather than actual data. Graph of the transformations to produce one RDD is called as Lineage Graph. If we lose a partition we can replay the transformation on that partition in lineage to achieve the same computation.
4. Lazy evaluation: Spark computes RDDs lazily the first time they are used in action so that it can pipeline transformations.
5. Persistence: Users can indicate which RDDs they will reuse and choose a storage strategy for them.
6. These properties of RDDs make them useful for fast computations.

Spark can run as local mode or stand-alone mode or in the cloud. Spark supports a stack of libraries including SQL and DataFrames in SparkSQL, MLlib for Machine Learning, GraphX for Graph Computing besides Spark Streaming.

DataFrame is the most important abstraction in SparkSQL's API. [12] It is a distributed collection of rows all having the same schema. Dataframes offer the relational power of SQL but with a functional programming model. Dataframes thus provide the best of the two worlds, sql and NoSQL. DataFrames in Spark are synonymous to tables in a relational database, and can be worked on like RDDs with various relational operators, such as where and groupBy. The difference between DataFrames and RDDs are that DataFrames are aware of their schema and keep track of same. Spark DataFrames are lazy not eager, a DataFrame object is a logical plan to compute a dataset, but execution occurs only when the user calls a save output operation. DataFrames can be constructed from tables in a system catalog or from existing RDDs of Java/Python objects. [12]

Spark SQL can be used to run advanced analytics algorithms on large clusters, including platforms for iterative algorithms [4] and graph analytics [3].

Spark SQL's User Defined Functions can be compared with Postgres User Defined Functions but in SparkSQL they can be full-fledged Spark programs. SparkSQL is a library on top of Spark, it exposes SQL interfaces via JDBC API, UDF or command line. It also has a Catalyst optimizer.

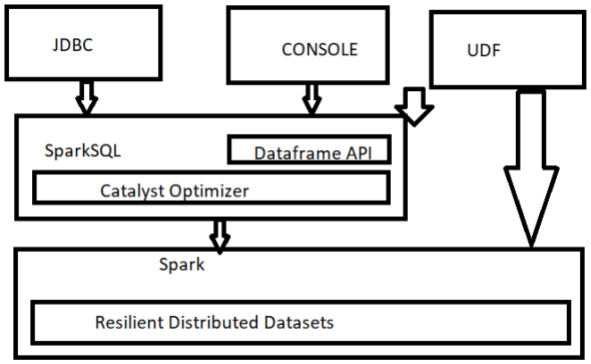

Figure2: Spark STACK shows the Spark architecture [6]

DataFrame operations in SparkSQL are supported by a relational optimizer Catalyst. Catalyst is a library that offers a framework to transform tree like structure and applying rules or functions to manipulate them. Catalyst supports both rule-based and cost-based optimization. Catalyst uses standard features like pattern matching, expressions and logical query plan [12]

2.2 Spatial Analysis Algorithm

2.2.1 Component Tasks

Component tasks implemented are the following:

2.2.1.1 Range Query using ST_Contains function

Given a query rectangle R and a set of points P, find all the points within R. The ST_Contains function takes a Rectangle and Point value.

Rectangle points are split into four using the String split function. Minimum x coordinates and minimum y coordinates are found using math min and max functions.

Point string is also split into two using the String split function. If the first point falls between the minimum and maximum x coordinates and the second point including boundary points falls between the minimum and maximum y coordinates, then the point is contained within the Rectangle.

2.2.1.2 Range Join Query function using ST_Contains function

Use ST_Contains function. Given a set of Rectangles R and a set of Points S, find all (Point, Rectangle) pairs such that the point is within the rectangle.

2.2.1.3 Distance query using ST_Within function

The ST_Within function takes two points Point A, Point B and a distance String.

Euclidean distance, the length between two points is calculated as

$$d=\sqrt{((x2 – x1)^2 + (y2 – y1)^2)} \quad (1)$$

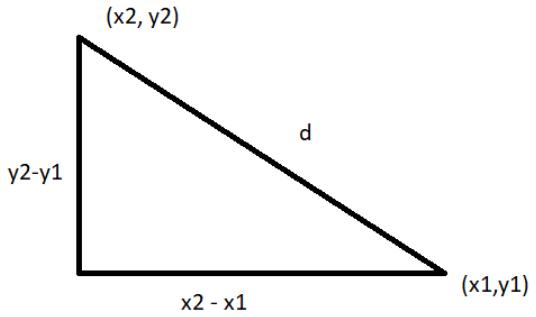

Figure 3: Euclidean distance between points [7]

If the Euclidean distance between the points is within the given distance the ST_Within function returns true.

The distance query function calls the ST_Within function to calculate Euclidean distance between the two points.

2.2.1.4 Distance join query using ST_Within function

Use ST_Within. Given a set of Points S1 and a set of Points S2 and a distance D in km, find all (s1, s2) pairs such that s1 is within a distance D from s2 (i.e., s1 belongs to S1 and s2 belongs to S2).

The distance range query function does a select from point p1, point p2 where ST_Within function is used to calculate Euclidean distance between the two points. A **spatial join** combines two or more datasets with respect to a spatial relationship.

2.2.1.5 HotZone

In Hotzone Analysis, we load pointPath and parse point data formats. Rectangle data is also loaded and the two datasets are joined making use of the ST_Contains function to join two datasets. This task performs a range join operation on rectangle datasets and point dataset. For each rectangle, the number of points located within the rectangle will be obtained. The hotter rectangle means that it includes more points. Hence this task is to calculate the hotness of all the rectangles.

2.2.1.6 HotCell

This task applies spatial statistics to spatio-temporal big data in order to identify statistically significant spatial hot spots using Apache Spark. [14]

Input: A collection of New York City Yellow Cab taxi trip records spanning monthly data say January 2009. The source data is an envelope encompassing the five New York City divisions in order to remove some of the noisy error data (e.g., latitude 40.5N – 40.9N, longitude 73.7W – 74.25W).

Output: A list of the fifty most significant hot spot cells in time and space as identified using the Getis-Ord statistic. This statistic will let us know where features with either high or low values cluster lie spatially.

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

where x_j = attribute value of cell j,
$w_{i,j}$ = spatial weight between cell i and j,
n = total number of cells,

The Getis-Ord statistic is also called a Z Score.
Zscore=
(sum of neighbor trips – mean * number of neighbors)/
(standard deviation *
Math.sqrt((n * number of
neighbors – number of neighbors * number of
neighbors)/(n-1))) (2)

The space-time cubes in our algorithm can be thought of as data cubes.

The neighborhood for each cell in the space-time cube is calculated by the neighbors in a grid. This is based on subdividing latitude and longitude uniformly.

This spatial neighborhood is created for the preceding, current, and following time periods (i.e., each cell has 26 neighbors). For simplicity of computation, the weight of each neighbor cell is presumed to be equal.

Algorithm 1: HotZone Analysis

- Load the data from the csv file and create a view
- Get the relevant columns from the data by querying the view in point 1
- Create a view ‘point’ with data in point 2
- load data from another CSV that contains the co-ordinates of rectangles and create a view ‘rectangle’
- Define and register a function ‘ST_Contains’ that returns bool for a given set of rectangle and point coordinated
- Do a join on the ‘rectangle’ and ‘point’ tables where ST-contains is True
- Do a groupby(rectangle) and count on the result to get the HotSpots on given data

Algorithm 2: HotCell Analysis

- Read data from the csv file, get the columns needed for us, the x,y co-ordinates and the date. Create a view ‘nyctaxitrips’ from data frame ‘pickupinfo’
- Calculate the boundaries on x,y and z based on the requirements
- The required can be re-written as zscore = (sum_of_neighbours_trips - mean * number_of_neighbours) / (standard deviation * Math.sqrt((num_of_cells * number_of_neighbours - number_of_neighbours * number_of_neighbours) / (num_of_cells - 1)))
- Mean = pickupInfo.count() / numCells
- standard deviation = math.sqrt(temp_top / numCells - (xbar * xbar)); where temp_top = spark.sql("select sum(count*count) from tripscount").first().get(0).toString.toDouble
- Register a function ‘getNeighbours’ to get all the surrounding cells of the give cell
- Register a function ‘checkneighbours’ that returns a bool, checks if the cell is with the boundaries of the requirement specified
- sumofNeighbours = spark.sql("select tc1.x, tc1.y, tc1.z, sum(tc2.count) as neighCountReduced, getNeighbours(tc1.x, tc1.y, tc1.z) as neighCount from tripscount tc1 CROSS JOIN tripscount tc2 where checkneighbours(tc1.x, tc1.y, tc1.z, tc2.x, tc2.y, tc2.z) group by tc1.x,tc1.y,tc1.z")
- Create and register a function get zscore that takes neighCountReduced and neighCount from the table query and returns the zScore of the cell.

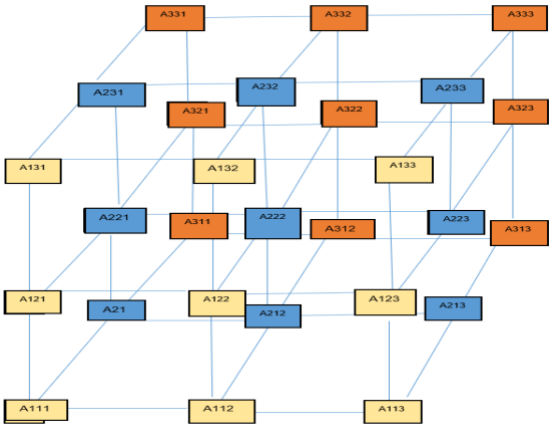

Figure 3: Datacube

3. EXPERIMENTAL SETUP AND COMPUTATIONAL DETAILS

3.1 Dataset

Dataset is mainly Point Data and Zone Data.

1. Point data: The input point dataset is the pickup point of New York Taxi trip datasets. The data format of this phase is from Data Systems Lab S3 Bucket[15]
2. Zone data is used only for hot zone analysis: at "src/resources/zone-hotzone" of the project template [15]

3.2 Input data- Hot zone analysis

The input point data is a small subset of NYC taxi dataset.

3.3 Input data -Hot cell analysis

The input point data is a monthly NYC taxi trip dataset (2009-2012) like "yellow_tripdata_2009-01_point.csv". The input data is about 2.5 GB.

3.4 Environmental Setup

During Phase I we setup clusters on our local machines and established one master and two slave configuration. During this setup we installed java 1.8 on all the machines and Hadoop 2.6.5 along with its compatible spark 2.2.0 version. After this we setup password less ssh so that master and slaves can be accessed from master node as shown in the figure 4.

For Phase II and III we setup similar configuration on tothlab but much more powerful with 32GB RAM, 3 CPU and disk 250 GB for faster query execution. Finally we tested the queries on our local machines as well as on tothlab cluster to compare the performance and other metrics.

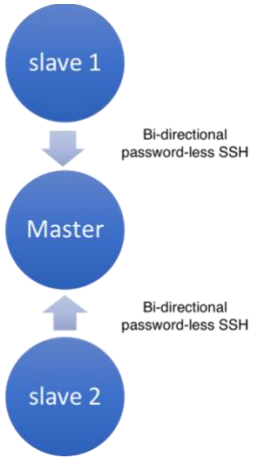

Figure 4: Master slave configuration

We setup Hadoop 2.6.5(one Master and two slaves) and Spark 2.2.0 in Thoth lab for running experiments.

We created a folder ‘test’ in Hadoop environment and uploaded input dataset yellow_tripdata_2009-01_point.csv to /test/output. [16]

3.5 Cluster Settings

1.Operating system: Ubuntu 14.04.1 LTS 32-bit
2.JDK 1.8.0
3. Hadoop 2.6.5
4.Spark 2.2.0
5.Passwordless bidirectional SSH

Commands for data setup:

./hdfs dfs -mkdir /test (3)

./hdfs dfs -mkdir /test/output (4)

./hdfs dfs -put ~/Downloads/ yellow_tripdata_2009-01_point.csv /test (5)

./hdfs fs -ls /test/output (6)
./hdfs fs -cat /test/output/yellow_tripdata_2009-01_point.csv (7)
./spark-submit jar file test/output hotcellanalysis hdfs://master:54310/test/ yellow_tripdata_2009-01_point.csv (8)

3.6 Output data format

3.6.1 Hot zone analysis

All zones with their count, sorted by "rectangle" string in an ascending order.

3.6.2 Hot cell analysis

The coordinates of top 50 hottest cells sorted by their G score in a descending order.

-7399,4075,15
-7399,4075,29
-7399,4075,22

3.7 Evaluation Metrics

Evaluation metrics analyzed are listed below

3.7.1 Execution time

SPARK provides much granular information about each stage of our Spark Job. Spark History Server is where we study the performance of Spark jobs like execution time.

3.7.2 Throughput

Throughput is defined as the amount of data passing through the SPARK system.

Higher the throughput the better as it stands for the aggregated progress of all jobs under co-location execution over running each job one by one using isolated execution.

3.7.3 Memory/core

This evaluation metrics shows how the memory footprint changes as the input size varies.

Iterations on same data

Iterations are run on same data until accuracy is improved.

Amount of data shuffled over network

Amount of data shuffled over network is another important metrics.

Selectivity Of Queries & Runtime

Selectivity factor of queries and runtime metrics are also evaluated.

3.7 System parameters

3.8.1 Hardware

Master 32GB RAM, 3 CPU, Disk 250 GB
Slave1 32GB RAM, 3 CPU, Disk 250 GB
Slave2 32GB RAM, 3 CPU, Disk 250 GB

3.8.2 Software

Spark2.2 is running in cluster mode.

4 RESULTS AND DISCUSSION

4.1 Execution time in seconds

4.1.1 Phase 2

For Phase II we executed the queries in the local cluster and the tothlab cluster gradually increasing the number of slaves and in turn we can see in figure 6 that the query execution time was reduced to 176,181,192,199 seconds for 9,6,4,2 cores respectively with increase in cpu cores.

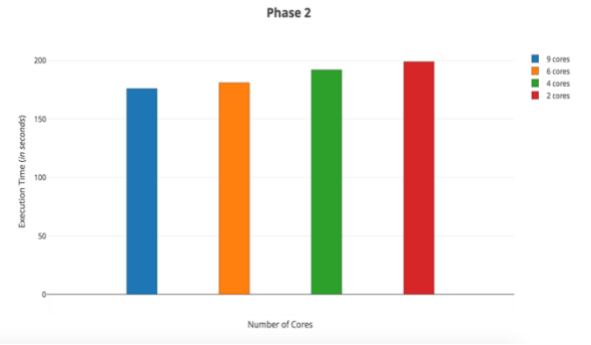

Figure 6: Execution time vs cpu cores for Phase II

4.1.2 Phase 3

4.1.2.1 Hot Zone Analysis

We tested Hot Zone with the similar setup as Phase II and the execution 27,29,33,47 seconds for 9,6,4 and 2 cores with similar trend as Phase 2.

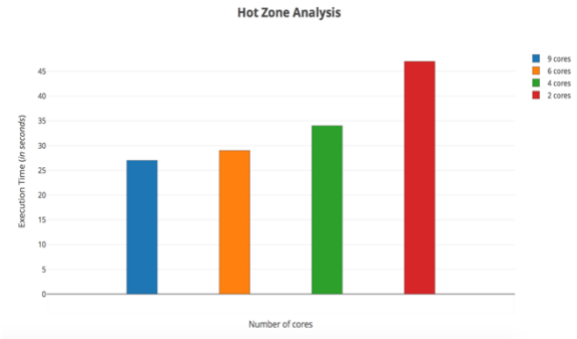

Figure 7: Execution time vs cpu cores for Hot Zone Analysis

4.1.2.1 Hot Point Analysis

We had to optimize the algorithm to secure this execution time. The figure 7 shows that increasing the number of cores has a noticeable impact on the execution time.

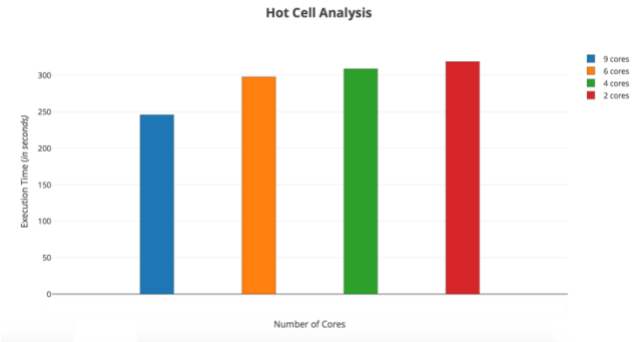

Figure 8: Execution time vs cpu cores for Hot Cell Analysis

4.2 Throughput

In the figure 9, 10 and 11 we can see that there is not much change.

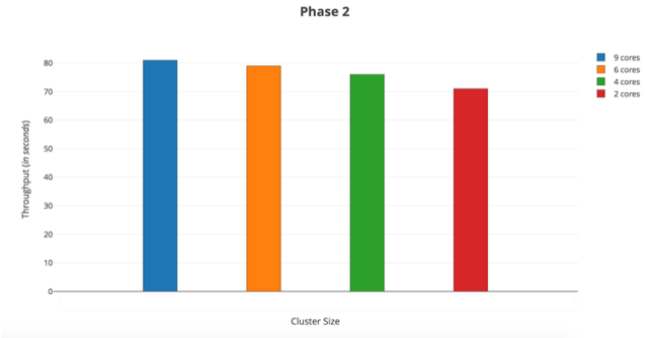

Figure 9: Throughput for Phase 2 for 4 Jobs

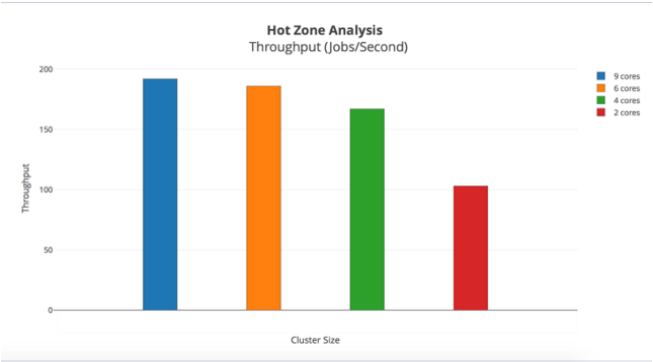

Figure 10: Throughput for Hot Zone for 12 Jobs

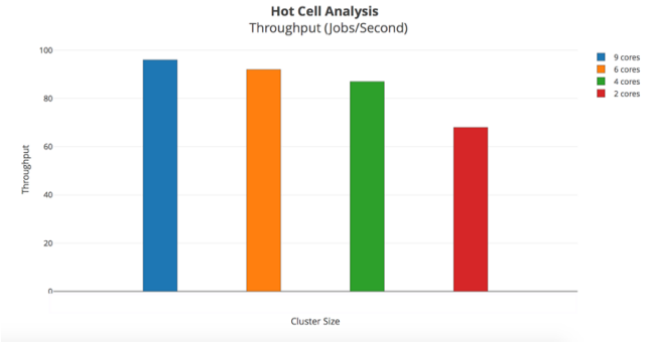

Figure 11: Throughput for Hot Point for 12 Jobs

4.3 Memory/core

The below figures show that memory usage on each node is less while increasing the number of executors for all the three tasks

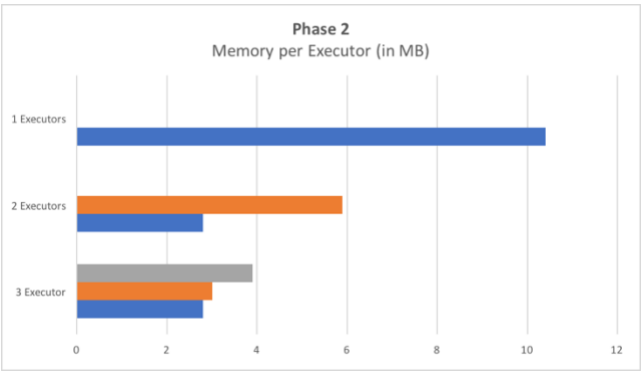

Figure 12: Memory per Executor vs Job for Phase II

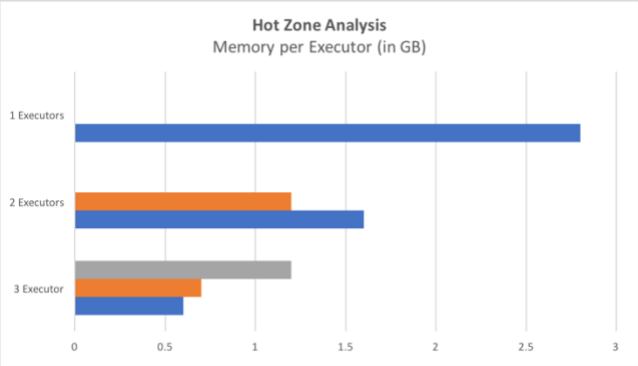

Figure 13: Memory per Executor vs Job for Hot Zone

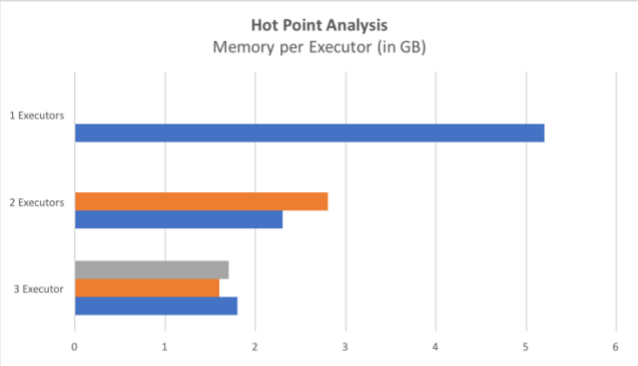

Figure 14: Memory per Executor vs Job for Hot Point

5 CONCLUSIONS

In summary, we have performed both an experimental and theoretical study of Spark. The experimental results have been used to successfully interpret the spatial profiles. RDDs offer an API based on coarse grained transformations that lets them recover data efficiently using lineage. RDDs implemented in Spark outperforms Hadoop by up to 20 times in iterative applications and can be used interactively to query large data.

SparkSQL is a brilliant module introduced in Spark to combine relational processing and complex analytics. SparkSQL extends Spark with a DataFrame API that is declarative in nature. SparkSQL is supported by an extensible Catalyst optimizer.

ACKNOWLEDGMENTS

Thanks to Dr Mohamed Sarwat, ASU & Ph.D. Student Yuhan Sun, ASU for their invaluable recommendations with the Hotspot Analysis.

REFERENCES

[1] Geospatial Analytics for Big Spatiotemporal Data: Algorithms, Applications, and Challenges∗-Ranga Raju Vatsavai † and Budhendra Bhaduri Computational Sciences and Engineering Division Oak Ridge National Laboratory, Oak Ridge, TN 37831.

[2] https://earthdata.nasa.gov/

[3] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. GraphX: Graph processing in a distributed dataflow framework. In OSDI, 2014.

[4].M. Zaharia et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In NSDI, 2012.

[5]Geospatial Analytics For Big Spatiotemporal data:Algorithms,Applications & Challenges http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/raju-bigspatial.pdf

[6] http://spark.apache.org/

[7] https://databricks.com/

[8] Discretized Streams:Fault Tolerant Streaming Computation at Scale- Matei Zaharia, T Das, H Li University of California, Berkeley

[9]http://onlinelibrary.wiley.com/doi/10.1111/j.1538-4632.1995.tb00912.x/epdf

[10] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5142189/

[11] https://en.wikipedia.org/wiki/Euclidean_distance

[12] https://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf

[13] http://sigspatial2016.sigspatial.org/giscup2016/problem

[14] Ord, J.K. and A. Getis. 1995. "Local Spatial Autocorrelation Statistics: Distributional Issues and an Application" in *Geographical Analysis* 27.

[15] https://datasyslab.s3.amazonaws.com/index.html?prefix=nyctaxitrips/

[16] https://www.thothlab.org/

[17] Getis, A. and J.K. Ord. 1992. "The Analysis of Spatial Association by Use of Distance Statistics" in *Geographical Analysis* 24(3).

[18] "GIS CUP 2016". http://sigspatial2016.sigspatial.org/giscup2016/problem.

[19] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, page 70. ACM, 2015.