Triangle Counting using Map/Reduce

Rathna Sindura Chikkam/team 9

CSE 6331, Spring 2019

Email: rathnasindura.Chikkam@mavs.uta.edu

References

- Suri, Siddharth & Vassilvitskii, Sergei. (2011). Counting triangles and the curse of the last reducer. Proceedings of the 20th International Conference on World Wide Web, WWW 2011. 607-614. 10.1145/1963405.1963491.
- https://csci8980bigdataalgo.files.wordpress.com/2013/09/lecture_01_sergei-vassilvitskii_intro-and-model.pdf
- http://archive.dimacs.rutgers.edu/Workshops/Parallel/slides/suri.pdf
- https://www2.cs.duke.edu/courses/fall15/compsci590.4/assignment3/MapReduceTutorial.pdf
- https://snap.stanford.edu/class/cs341-2013/downloads/amazon-emr-tutorial.pdf
- http://megasthenis.github.io/repository/Hadoop-WNCG-Intro.pdf
- https://en.wikipedia.org/wiki/Clustering_coefficient
- https://en.wikipedia.org/wiki/Social_network
- https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
- https://www.jpytr.com/post/analysinggeographicdatawithfolium/

Objective, Problem Statement

- Objective of the project
  - To implement the NodeIterator++ algorithm proposed by the authors using both sequential and Map/Reduce version of algorithms and compare the efficiency of both the approaches.

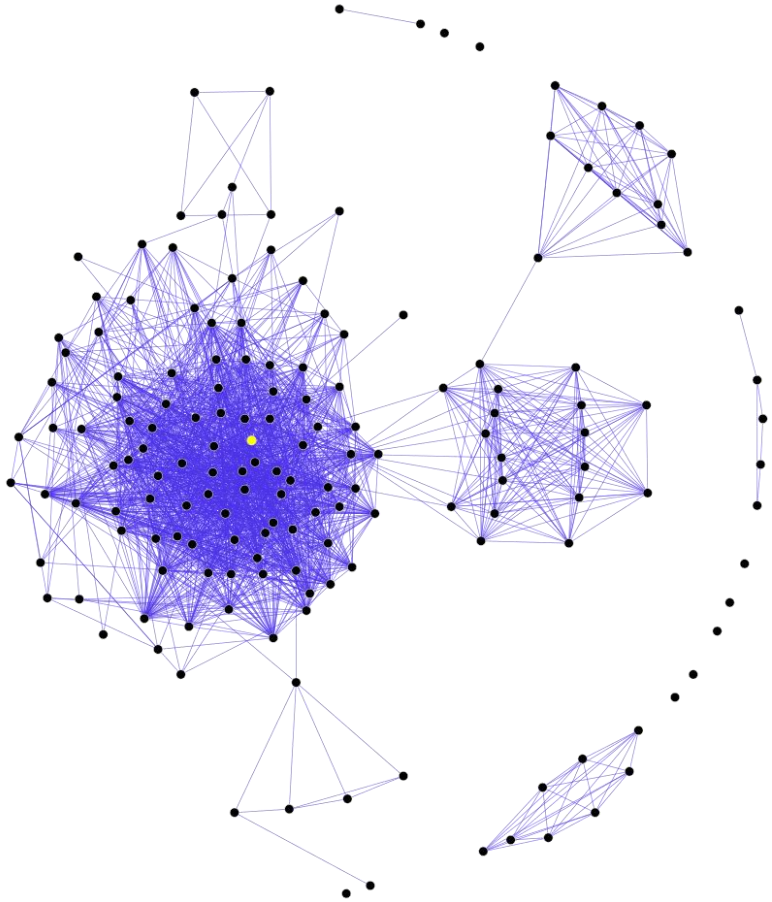
- Problem Statement

Clustering coefficient of a node in a social network :

$$cc(v) = \frac{|\{(u,w) \in E \mid u \in \Gamma(v) \text{ and } w \in \Gamma(v)\}|}{\binom{d_v}{2}}.$$

  - cc(v) can be computed by counting the number of triangles incident on the particular node in the network.
  - When the network is huge, we would need a computing system that can still calculate the clustering coefficient in as much less time as possible without running into any memory issues. This calls for the use of Parallel Computing of the graph data, hence the use of Map/Reduce method of programming.

Approach, Datasets to be used

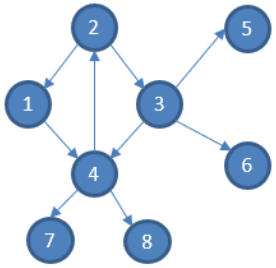


Sample Social Network Graph

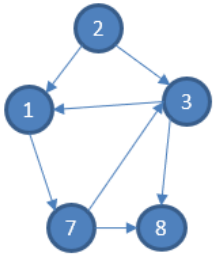
- Approach
  - To implement a sequential version of proposed NodeIterator++ algorithm
  - To implement a Map/Reduce version of proposed algorithm
  - Compare the efficiency of the implementations using various datasets
- Data sets used
  - UK Accident Graph
  - Dataset from SNAP Stanford website

For Initial Presentation

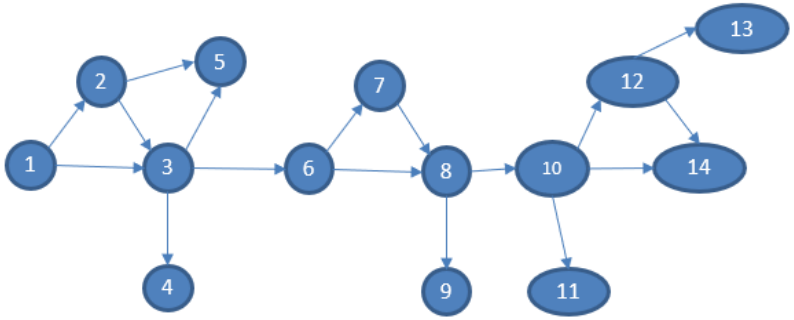
- Implemented the NodeIterator++ algorithm in python.
- Created sample graph data with 10-20 nodes, pictorial representation of some of the graphs used and the algorithm output are as below:



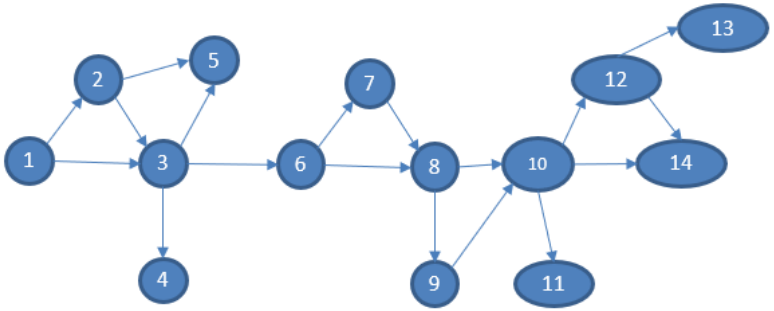
Graph1, Triangle_Count: 2



Graph2, Triangle_Count: 2



Graph3, Triangle_Count: 4



Graph4, Triangle_Count: 5

What is MapReduce and why MR approach?

- "Simple but very powerful computational framework specifically designed to enable fault-tolerant distributed computation across a cluster of centrally managed machines."*

Why MR?

- Because of the framework's ability to perform parallel computation across clusters, authors chose MR to improve their sequential algorithm.

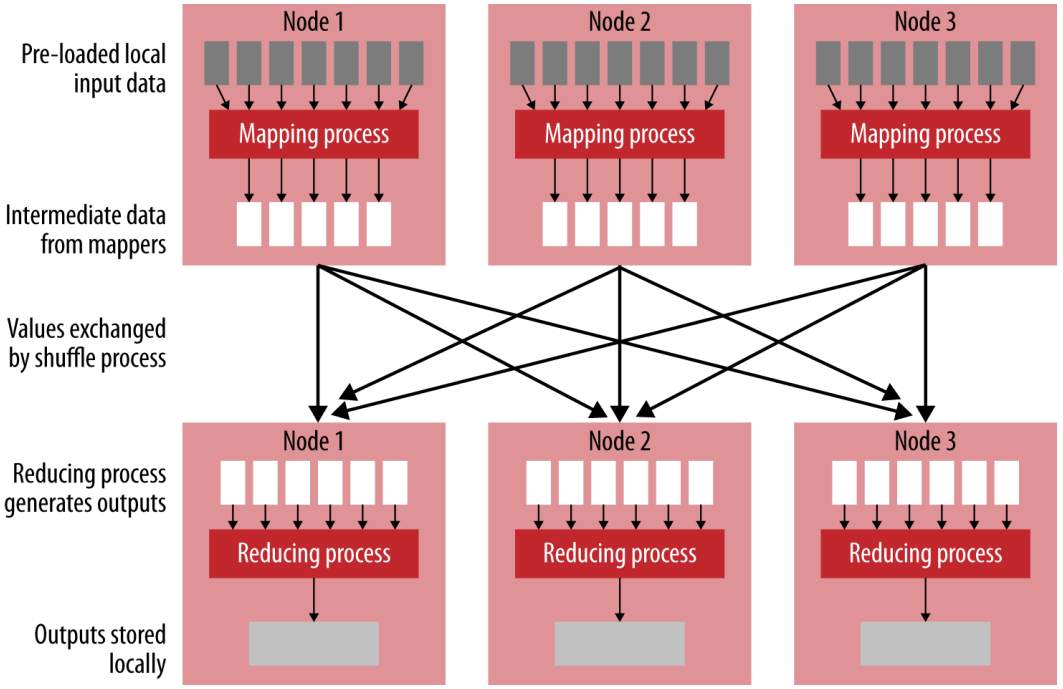


Fig.: Data flow of a MapReduce job being executed on a cluster of a few nodes

* – from Data Analytics with Hadoop by Benjamin Benfort, Jenny Kim

Details of theoretical results

Differences between sequential and MR algorithms

Sequential

Algorithm 2 NodeIterator++(V,E)

1: $T \leftarrow 0$;
2: **for** $v \in V$ **do**
3: **for** $u \in \Gamma(v)$ and $u \succ v$ **do**
4: **for** $w \in \Gamma(v)$ and $w \succ u$ **do**
5: **if** $((u, w) \in E)$ **then**
6: $T \leftarrow T + 1$;
7: **return** T;

Complexity: $O(m^{3/2})$
With 3 for loops, we can know the triangle count for 5000 nodes data in 98 hrs. approx. !!!

MapReduce

Algorithm 3 MR-NodeIterator++(V,E)

1: **Map 1:** Input: $\langle (u, v); \emptyset \rangle$
2: **if** $v \succ u$ **then**
3: emit $\langle u; v \rangle$
4: **Reduce 1:** Input $\langle v;\ S \subseteq \Gamma(v) \rangle$
5: **for** $(u, w) : u, w \in S$ **do**
6: emit $\langle v; (u, w) \rangle$
7: **Map 2:**
8: **if** Input of type $\langle v; (u, w) \rangle$ **then**
9: emit $\langle (u, w); v \rangle$
10: **if** Input of type $\langle (u, v); \emptyset \rangle$ **then**
11: emit $\langle (u, v); \$ \rangle$
12: **Reduce 2:** Input $\langle (u, w); S \subseteq V \cup \{\$\} \rangle$
13: **if** $\$ \in S$ **then**
14: **for** $v \in S \cap V$ **do**
15: emit $\langle v; 1 \rangle$

Complexity: O(n)

Output of the algorithms - comparison

- As can be seen below, the map-reduce algorithm performed quite effectively in contrast to its sequential counterpart for both the accident data that were either effect by various light conditions, or by various weather condition.

Dataset	Node Size	Edges Count	Sequential Results		MR Results	
			Triangle Count	Execution Time (Sequential) (in sec)	Triangle Count	Execution Time (MR) (in sec)
Light	100	61	3	0.140350103	15	0.00799942
	500	2109	10806	4.628014088	1555	1.084049702
	1000	8172	97889	188.3723097	6676	15.62057304
Weather	100	98	8	0.012014866	30	0.007998705
	500	2740	21877	9.43204093	2130	1.691673517
	1000	10381	178693	398.0261416	8691	25.2680366

Complexity: $O(m^{3/2})$

Complexity: O(n)

- Also, another important thing to notice here is the **Triangle Count being different for both versions** of the algorithm. Reason? MR Algorithm doesn't take into consideration of node degree of all the noticed triangles. Interesting.

Accident Data – Light condition comparison

Below is a comparison of top 20 values of light data on 100, 500 and 1000 nodes whose triangle count on each vertex is provided. Turns out, no matter the amount of dataset taken to calculate the triangle count, Light Condition 1 seems to be the reason for accidents on majority of nodes. The triangle count data is obtained by using **nodeiterator++** algorithm (sequential version)

Light_100			Light_500			Light_1000		
Vertex	Triangles on Vertex	Light Condition on vertex	Vertex	Triangles on Vertex	Light Condition on vertex	Vertex	Triangles on Vertex	Light Condition on vertex
57	2	4	497	5575	1	999	51761	1
59	2	1	499	5575	1	997	51726	4
61	2	1	489	5566	1	993	51725	1
63	2	1	491	5566	4	995	51725	6
65	2	1	493	5566	4	985	51698	1
67	2	1	495	5566	4	987	51698	1
69	2	1	487	5560	1	989	51698	1
71	2	1	485	5471	1	991	51698	1
73	2	1	479	5467	1	981	51628	1
75	2	1	481	5467	1	983	51628	1
77	2	4	483	5467	4	979	51623	1
79	2	1	500	5231	4	977	50186	1
81	2	4	498	5222	1	975	50177	4
83	2	1	496	5214	1	971	50176	1
85	2	1	494	5206	1	973	50176	4
87	2	1	469	5202	1	969	48985	1
89	2	1	471	5202	4	965	48950	1
91	2	1	473	5202	1	967	48950	1
93	2	4	475	5202	1	957	48949	4
95	2	1	477	5202	1	959	48949	1

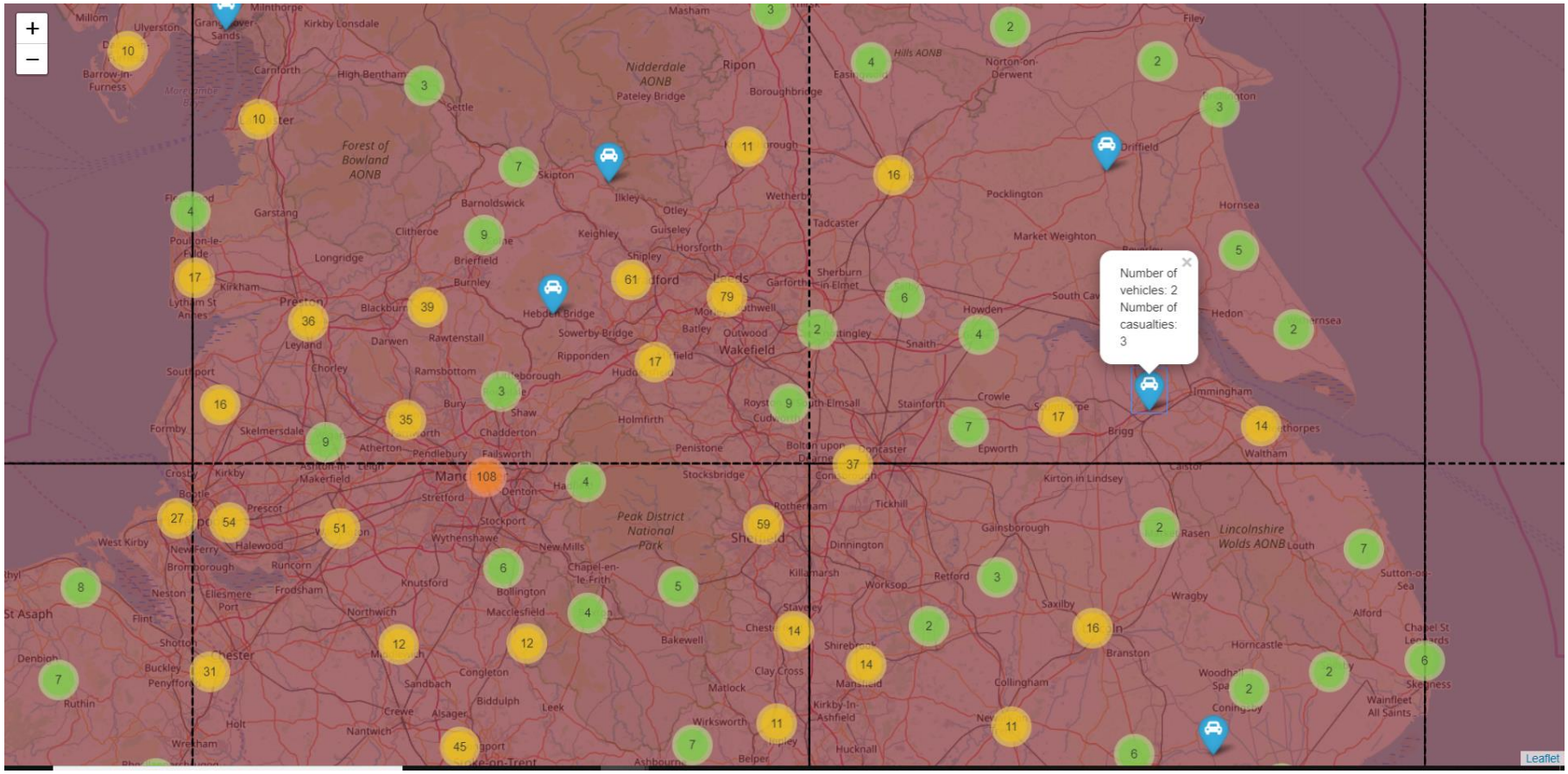
Accident Data – Weather condition comparison

Below is a comparison of top 20 values of weather data on 100, 500 and 1000 nodes whose triangle count on each vertex is provided. Turns out, no matter the amount of dataset taken to calculate the triangle count, Weather Condition 1 seems to be the reason for accidents on majority of nodes. The triangle count data is obtained by using **nodeiterator++** algorithm (sequential version)

Weather_100			Weather_500			Weather_1000		
Vertex	Triangles on Vertex	Weather Condition on vertex	Vertex	Triangles on Vertex	Weather Condition on vertex	Vertex	Triangles on Vertex	Weather Condition on vertex
5	6	1	500	11939	1	999	90284	1
7	6	1	498	11906	1	997	90236	1
9	6	1	496	11900	1	993	90235	1
11	6	1	494	11894	1	995	90235	8
13	6	1	484	11422	1	985	90212	1
15	6	1	486	11422	9	987	90212	2
17	6	1	488	11422	1	989	90212	1
19	6	1	490	11422	1	991	90212	1
21	6	2	492	11422	1	981	90152	1
23	6	1	480	10853	1	983	90152	1
25	6	1	482	10853	2	979	90143	1
27	6	1	478	10710	1	998	88409	1
29	6	1	476	10709	1	1000	88409	1
31	6	1	462	10708	1	996	88396	1
33	6	1	464	10708	1	988	88388	1
35	6	1	466	10708	1	990	88388	1
37	6	1	468	10708	1	992	88388	1
39	6	1	470	10708	8	994	88388	1
41	6	1	472	10708	1	980	87980	1
43	6	1	474	10708	1	982	87980	1

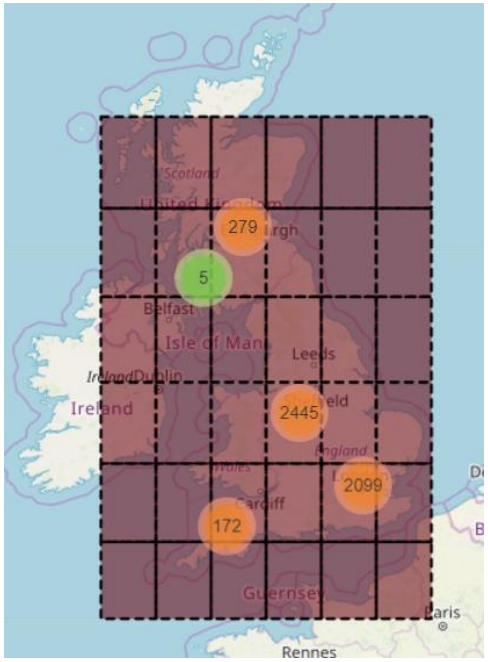
Accident Data – Light & Weather visual comparison

These comparisons are made by using the initial data to see geographically which areas of UK are affected by accidents due to which of Light or Weather conditions

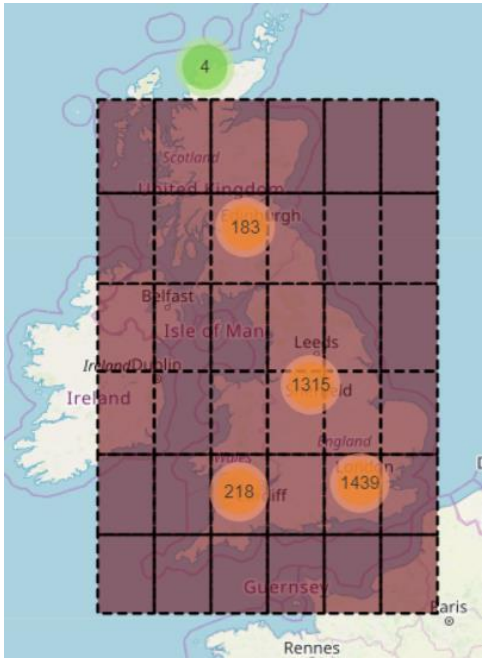


Accident Data – Light & Weather comparison

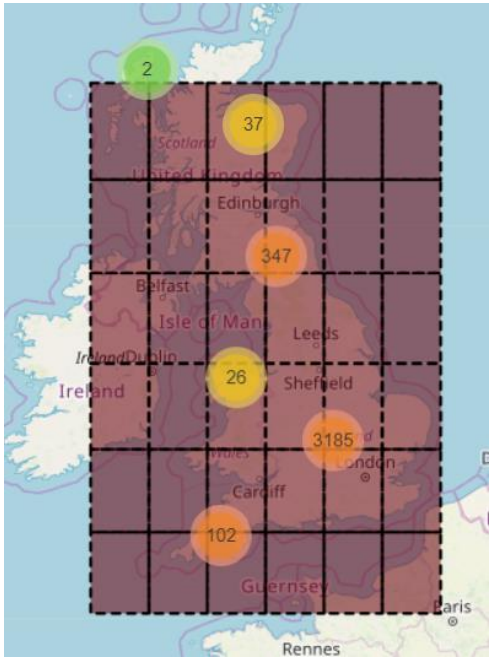
- Used Folium package available on python platform in order to analyze how much of UK is affected by either of poor light or weather conditions. The below are the results:



All the accident data

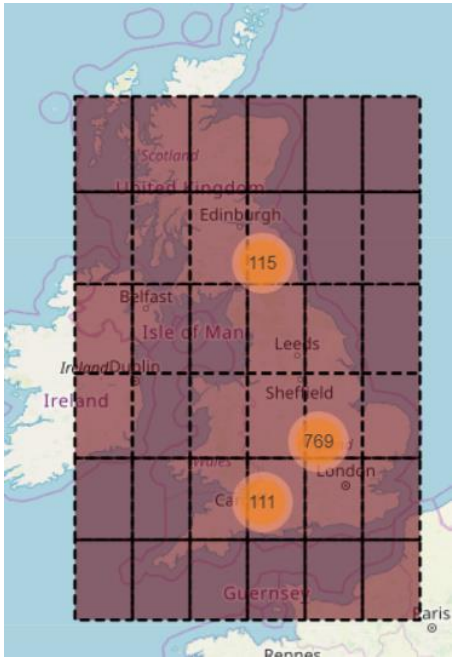


Data with same Light and Weather condition rating

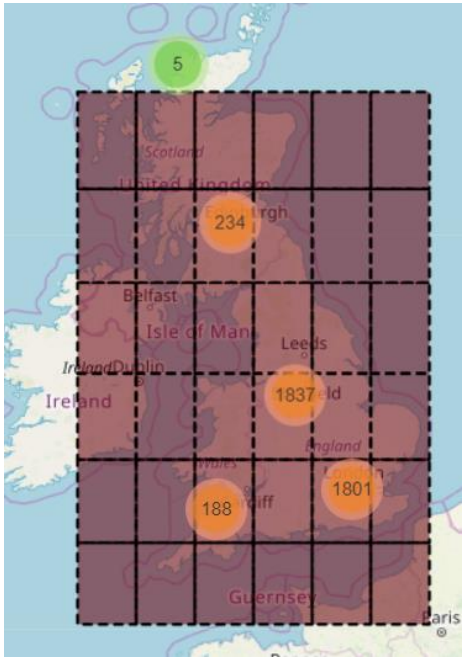


Data with 1st worst Light condition rating

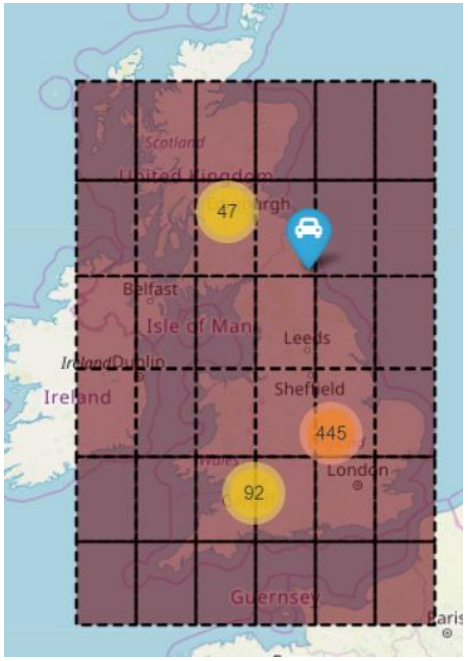
Accident Data – Light & Weather comparison



Data with 2nd worst Light condition rating



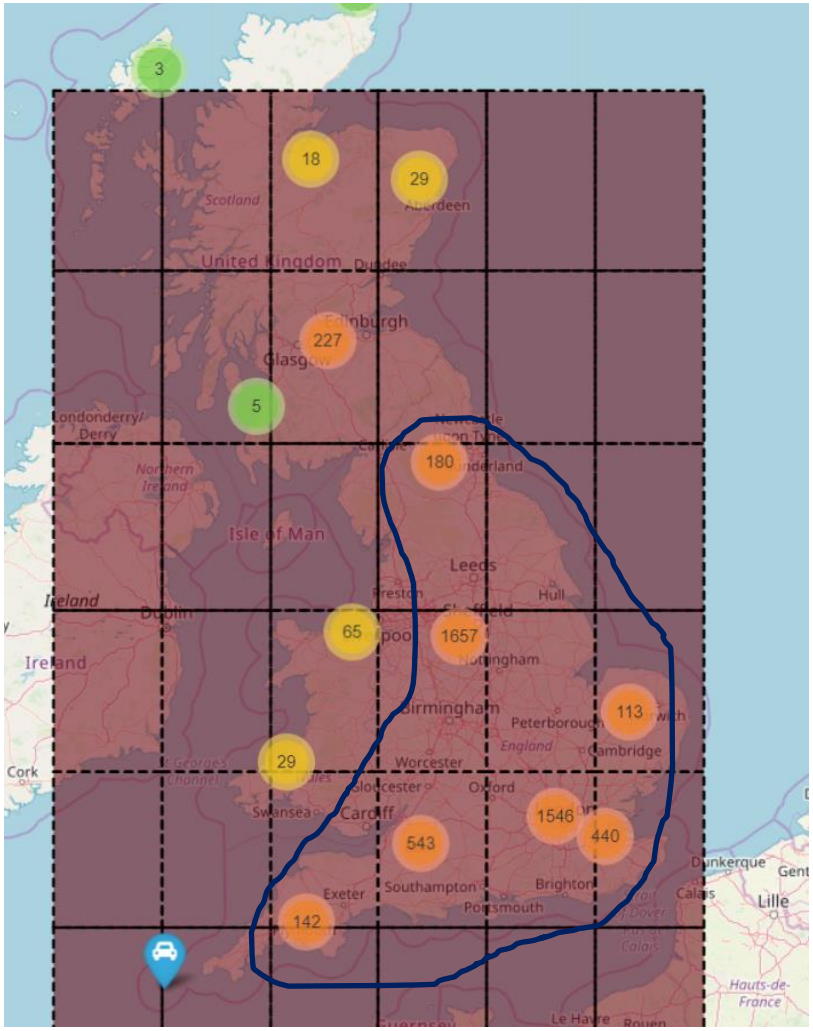
Data with 1st worst Weather condition rating



Data with 2nd worst Weather condition rating

Final Visual Analysis on dataset

- For both the light and weather data set, individually as well as when they have same rating, also the over all UK accidents dataset, they all seem to have high accidents in the mid east to south and south east region of UK, as clusters shown in the graphs tend to form a strong hub, as shown in the blue dotted area beside.



Evaluation

- Have you finished the project as per initial description?
  - Able to create and run both the algorithms locally and the results seemed impressive.
  - Able to run only 100, 500, 1000 nodes worth of data as the MRJobs created to run the mappers and reducers could not instantiate and run an AWS Elastic MapReduce cluster properly.

- Did you encounter any problems
  - It took quite some time to understand that the algorithms are supposed to provide different outputs
  - Whether using MRJob or trying to implement the Mappers and Reducers in GCP or AWS, there isn't much documentation available to know how to do them.
  - Referenced articles were of great help!

Conclusions

- What you think you learned from this effort
  - My takeaways are working with Hadoop, MapReduce, MRJob and visualizations for accident graph

- Though the MapReduce is a decent framework, not many articles are out there for implementing MapReduce algorithms in python and then execute them over Hadoop since Apache Spark Framework took over.

Thank You !!!