Sotera Aggregate Micro-path / Trip Line Analytic.

SPARK Implementation Notes

**SUMMARY**

The spark implementation of the trip line analytic is a highly flexible / extensible implementation meant to allow ease of use with multiple input formats. The analytic takes as input geo located time stamped data and outputs up sampled aggregated points where ever a path from the input data cross a “trip line”. The output is written as avro serialized tiles in using a mercator projection. These tiles can then be viewed using the Oculus Tile Sever.

Flexibility / extensibility of the analytic is provided in two ways. The first is the configuration file which allows you to set various parameters for the job. The second is in code, allowing you to extend the MicroPathEngine.scala class to specify your own input format.

**RUNNING**

We use SBT both to build and run the analytic. SBT is not required but simplifies things greatly. To run.

sbt clean package “run <your job conf file>”

**CONFIGURATION**

Different runnable classes can add or omit configuration options, the base options are described here.

|  |  |
| --- | --- |
| Option | Detail |
| master\_uri | The spark cluster master uri. For example spark://ip:port |
| SPARK\_HOME | Install location of spark on the cluster |
| deployment\_path | The location of the jar contacting your code, for spark to distribute. |
| default\_parallelism | The default level of parallelism for spark, recommend value is ~1.5 \* number of cores available. |
| job.name | Name of the spark job |
|  |  |
| micropath.engine | The class to load which executes the trip line analytic. See below |
| input.path | Path to find input data |
| **output.path** | **Path to save results** |
| col.id | Index of the id column in the input data |
| col.datetime | Index of the datetime col in the input data, or -1 if col.data and col.time are used. See runnable class for details. |
| col.date | Index of the date col in the input data, or -1 if col.datetime is used. |
| col.time | Index of the time col in the input data, or -1 if col.datetime is used. |
| col.lon | Index of the longitude column (Double) in the input data. |
| col.lat | Index of the latitude column (Double) in the input data. |
| col.seperator | Field delimiter in the input data (defaults to hive default delimiter if not set). |
|  |  |
| mercator.level | The highest level you wish to produce. Output will consist of levels 1 -> N |
| avro.data.store | “hdfs” or “hbase” |
| avro.output.name | The name of the output. |
| avro.output.desc | An optional description. |
|  |  |
|  |  |
| time.filter | Filter out points with delta time > time.filter (seconds) |
| velocity.filter | Filter out paths with velocity > velocity filter (km/hr) |
| distance.filter | Filter out points with delta distance > distance.filter (meters) |
| lower.lat | Lower most latitude to calculate over. (default -90) |
| upper.lat | Upper most latitude to calculate over. (default 90) |
| lower.lon | Lower most longitude to calculate over. (defaults to - 180) |
| upper.lon | Upper most longitude to calculate over. (defaults to 179.999999) |

**RUNNABLES**

You can extend or modify the analytic by extending the MicroPathEngine class and then specifying your new class in the micorpath.egine configuration option. Each of the current implementations are described below.