Paper SAS395-2017

Location Analytics: Minority Report Is Here—Real-Time Geofencing Using SAS® Event Stream Processing

Frederic Combaneyre, SAS Institute Inc.

ABSTRACT

Geofencing is one of the most promising and exciting concepts that has developed with the advent of the Internet of Things. Like John Anderton in the 2002 movie “Minority Report,” you can now enter a mall and immediately receive commercial ads and offers based on your personal taste and past purchases. Authorities can track vessels’ positions and detect when a ship is not in the area it should be, or they can forecast and optimize harbor arrivals. When a truck driver breaks from the route, the dispatcher can be alerted and can act immediately. And there are countless examples from manufacturing, industry, security, or even households. All of these applications are based on the core concept of geofencing, which consists of detecting whether a device’s position is within a defined geographical boundary. Geofencing requires real-time processing in order to react appropriately. In this session, we explain how to implement real-time geofencing on streaming data with SAS® Event Stream Processing and achieve high-performance processing, in terms of millions of events per second, over hundreds of millions of geofences.

INTRODUCTION

One of the most important underlying concepts behind all location-based applications is called **geofencing**. Geofencing is a feature of an application that defines geographical boundaries. A geofence is a virtual barrier. So, when a device enters (or exits) the defined boundaries, **an action is immediately triggered** based on specific business needs.

One of the early commercial uses of geofencing was in the livestock industry, where a handful of cattle in a herd would be equipped with GPS units and if the herd moved outside of geographic boundaries set by the rancher, the rancher would receive an alert.

What applies to the flow of cattle can also be applied to:

- Fleet management: When a truck driver breaks from his route, the dispatcher can be alerted and act immediately.
- Customs transport: Authorities can track vessels' positions and detect when a ship is not in the area it should be or forecast and optimize harbor’s arrivals.
- Public areas, like airports or train stations: The flow and density of people can be detected in real time in order to remove bottlenecks and optimize queuing times, adapt path guidance, organize staffing, or optimize flow path and procedures.
- Galleries and museums: Administrators can quantify the popularity of exhibits, identify under-used spaces, and use visitor behavior to optimize future events.
- Shopping centers: Geofencing can show in real time how many people pass in front of a certain store, shelf, information point, or door, and where these people are coming from. How many people are watching a certain TV ad on a billboard? Where is the best place to position a promotion based on foot traffic? Hence, geofencing allows optimizing store workflow (goods supply, cart management…) and layout.

And there are countless examples from manufacturing, industry, security, or even households—like an ankle bracelet alerting authorities if an individual under house arrest leaves the premises, or automatically switching lights off when the whole family leaves the house.

An important paradigm that is inherent to all those applications is **the immediacy of action**.

In order to react appropriately, the position information has to be processed immediately, with low latency, regardless of the volume of events it is required to analyze. Taking too much time to react is not an option in such cases, as the subject/device will already have moved to another location. Hence, a delayed reaction is obsolete.

Of course, a timely reaction is just one part of the game. We also need to **react appropriately**. Deciding the best action to apply often means being able to detect specific **complex events patterns** out of the masses of events and to **apply high-end analytics** or machine learning algorithms on real-time data streams. And this is where SAS® software like SAS® Event Stream Processing comes into play, providing high performance and low latency geofencing analysis and high-end streaming analytics, as well as real-time predictive and optimization operations.

Released in early 2017, SAS Event Stream Processing 4.3 introduces a new Geofence window that provides real-time geofencing analysis capabilities on streaming events. This Geofence window was already available as a custom additional plug-in for SAS Event Stream Processing 4.1 and 4.2 and is now totally integrated as a standard SAS Event Stream Processing window.

SAS EVENT STREAM PROCESSING GEOFENCE WINDOW

The SAS Event Stream Processing Geofence window allows determining if a location position coming from an event stream is inside a defined area of interest or close to a defined location of interest, and augmenting this event with these area or location details.

The Geofence window behaves functionally like a lookup or an outer join window, where the geofence areas and locations are on the dimension side and the event position on the streaming side. Hence, like a Join window, the Geofence window requires 2 input windows, one for injecting the geofence areas or locations, called geometries, and the other for the streaming events, called positions.

Figure **1** shows a sample Event Stream Processing streaming model implementing the Geofence window.

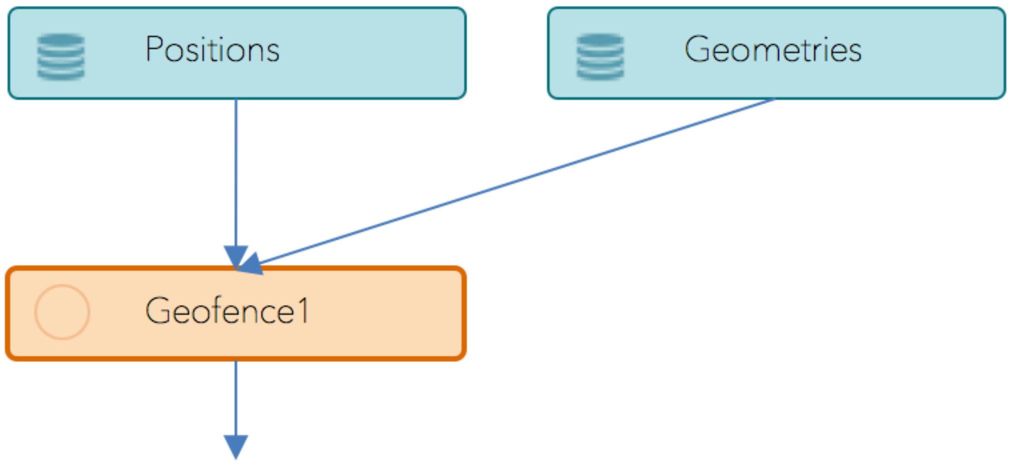

Figure 1**. Event Stream Processing Model Using the Geofence Window**

In an event stream processing XML model, the first window connected to the Geofence window is the position window. When using the SAS Event Stream Processing Studio, each window's role is defined in the property panel.

Areas and locations of interest are defined as geometry shapes. The Geofence window supports 2 types of geometries: **polygons** and **circles**.

Geometries are published as events, one event per geometry. The Geofence window supports insert, update, and delete opcodes, allowing dynamic update of the geometries.

The Geofence window is designed to support any coordinate type or space, either Cartesian or geographic. The only requirement is that all coordinates must be consistent and refer to the same space or projection. For geographic coordinates, the coordinates must be specified in the (X,Y) Cartesian order (longitude, latitude). All distances are defined and calculated in meters.

Let's cover now how the Geofence window implements the two types of geometries: polygons and circles.

POLYGON GEOMETRIES

A polygon is a plane shape representing an area of interest. The Geofence window supports polygons, multi-polygons, and polygons with holes or multiple rings.

Figure 2 below shows some sample polygon geofences.

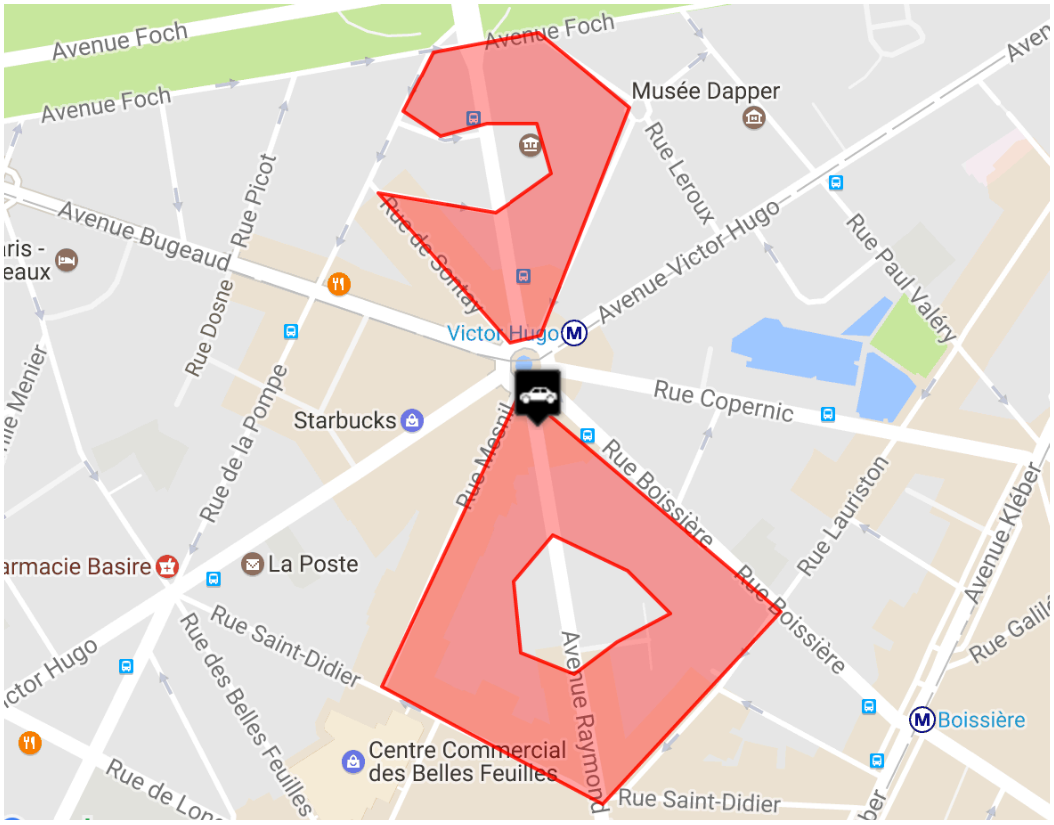

Figure 2. Sample Polygon Geofences

A polygon is defined as a list of position coordinates representing the polygon's rings. A ring is a closed list of position coordinates. In order to be considered closed, the last point of the ring list must be the same as the first one. So, for example, a ring that is geometrically defined with 4 points like a square must declare 5 position coordinates, the last point being the same as the first one.

The input polygon window schema must have at least the following 2 mandatory fields:

- A single **key** field of type **int64** or **string.** This field defines the ID of the geometry.

- A data field of type **string**. This field contains the list of the rings' position coordinates. The coordinates are defined as a list of numbers (double) separated by spaces in the X, Y order. For polygons with multiple rings, the first ring defined must be the exterior ring and any others must be interior rings or holes. For example, the following string represents a polygon made of 4 points that includes a hole made of 7 points:

"**5.281 9.455** 3.607 7.112 6.268 6.181 8.414 7.705 **5.281 9.455** 5.671 8.316 6.572 8.033 7.087 7.695 6.444 7.469 5.929 7.215 5.285 7.384 5.199 7.949 5.671 8.316"

If the polygon data is provided using the standard GeoJSON format, you can easily parse it and format it using a functional window.

The schema can also have an optional description field that can be propagated with the Geofence Window output event.

All other fields will be ignored.

When working with polygons, the Geofence window analyzes each event position coming from the streaming window and returns the polygon this position is inside of. If there are multiple matching geometries (in case of overlapping polygons) and if the option **`output-multiple-results`** is set to true, multiple events are produced (one per geometry).

The Geofence window behaves like a lookup join, so its output schema is automatically defined and includes all fields coming from the input position window appended with the following additional fields:

- A mandatory field of type int64 or string that will receive the ID of the geometry. If no geometries are found, the value of this field will be null in the produced event. This field is defined by the parameter **`geoid-fieldname`**.
- An optional field that will receive the description of the geometry if it exists in the geometry window schema. This field is defined by the parameter **`geodesc-fieldname`**.
- An optional field of type double that will receive the distance from the position to the centroid of the polygon. This field is defined by the parameter **`geodistance-fieldname`**.
- If **`output-multiple-results`** is set to `true`, a mandatory key field of type int64 that will receive the event number of the matching geometry. This field is defined by the parameter **`eventnumber-fieldname`**.

Below is a sample event stream processing XML model that implements a Geofence window using polygons:

```
<project name="geofencedemo" pubsub="auto" threads="4" index="pi_EMPTY">
 <contqueries>
  <contquery name="cq1" trace="alerts">
   <windows>
    <window-source name="position_in" pubsub="true" insert-only="true">
     <schema>
      <fields>
       <field name="pt_id" type="int64" key="true"/>
       <field name="GPS_longitude" type="double"/>
       <field name="GPS_latitude" type="double"/>
       <field name="speed" type="double"/>
       <field name="course" type="double"/>
       <field name="time" type="stamp"/>
       </fields>
     </schema>
    </window-source>
    <window-source name="poly_in" pubsub="true" insert-only="true">
      <schema>
```

```
          <fields>
            <field name="poly_id" type="int64" key="true"/>
            <field name="poly_desc" type="string"/>
            <field name="poly_data" type="string"/>
          </fields>
        </schema>
      </window-source>
      <window-geofence name="geofence_poly" index="pi_EMPTY">
        <geofence
              coordinate-type="geographic"
              log-invalid-geometry="false"
              output-multiple-results="false"
              autosize-mesh="true"
              max-meshcells-per-geometry="200"
              />
        <geometry
              data-fieldname="poly_data"
              desc-fieldname="poly_desc"
              data-separator=" "
              />
        <position
              x-fieldname="GPS_longitude"
              y-fieldname="GPS_latitude"
              />
        <output
              geoid-fieldname="poly_id"
              geodesc-fieldname="poly_desc"
              geodistance-fieldname="poly_dist"
              />
     </window-geofence>
     </windows>
     <edges>
      <edge source="position_in"   target="geofence_poly"/>
      <edge source="poly_in"       target="geofence_poly"/>
     </edges>
    </contquery>
  </contqueries>
</project>
```

CIRCLE GEOMETRIES

A circle defines the position of a location of interest. It is defined as a couple of coordinates (X, Y) or (longitude, latitude) representing the center of the circle and a radius distance around this position.

Figure 3 below illustrates some sample circle geofences.

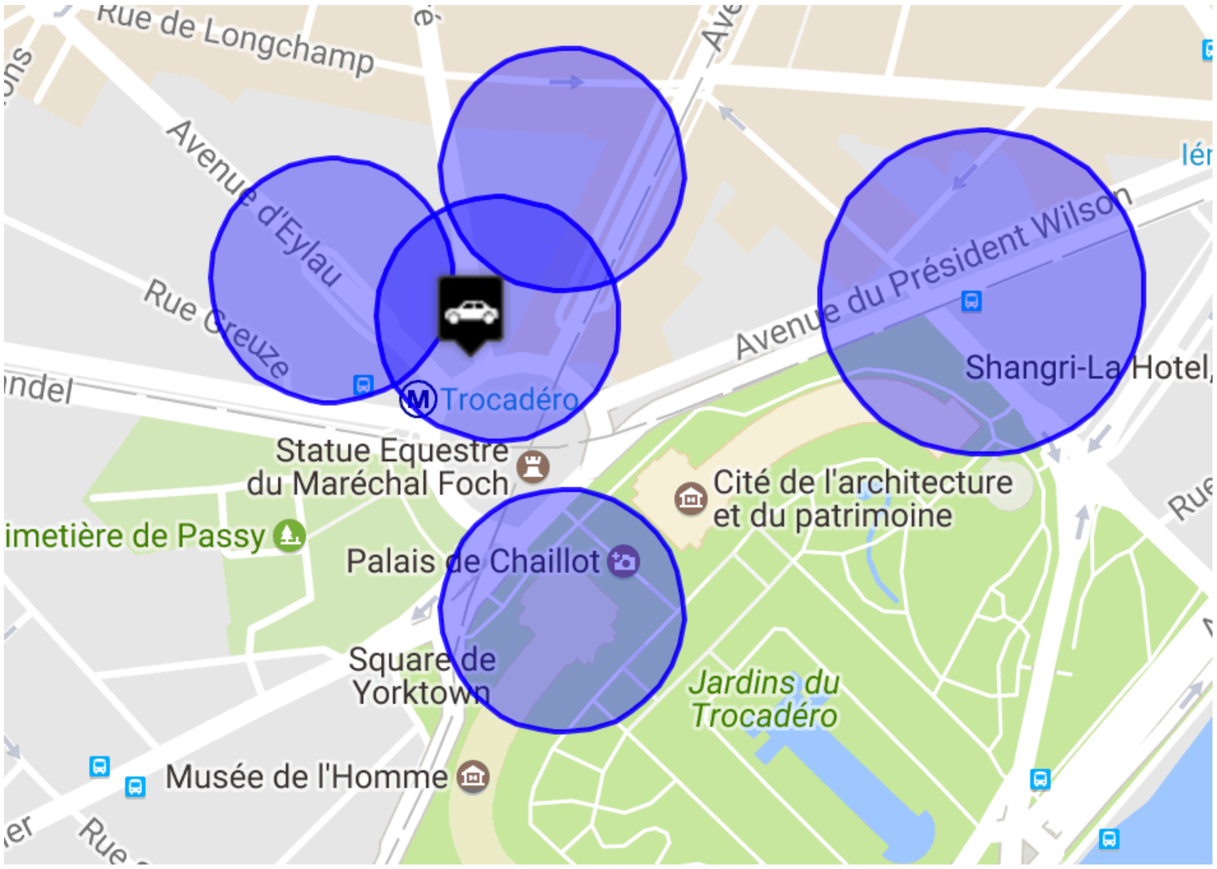

Figure 3. Sample Circle Geofences

The input circle geometry window schema must have at least the following 3 fields:

- A single **key** field of type **int64** or **string**. This field defines the ID of the circle geometry.
- 2 coordinate fields of type **double** that contain the X and Y coordinates of the circle center.

The schema can also have the following optional fields:

- A radius field of type **double**, representing a circular area around the center point position. If this field is not specified, the default distance defined by the parameter **`radius`** will be used.
- A description field that can be propagated with the Geofence Window output event.

All other fields will be ignored.

When working with circles, the Geofence window analyzes each event position coming from the streaming window and returns the circle ID, which matches the following criteria:

- If the position lookup distance is set to 0, then the position behaves like a simple point. It is either in or out of the circle. If it is in the circle, we have a match.
- Similarly, if the circle radius is set to 0, then the circle behaves like a bare point, and it is only required for this point to be within the position lookup distance area for having a match.
- For any other value of the position lookup distance and the circle radius, then the position and the circle's center have to be within each other's distance to have a match. It means that the position is within the circle and the distance between the circle's center and the position is lower than the lookup distance. Figure 4 below illustrates the circle's geometry lookup logic in such a case.

- And finally, if both the position lookup distance and the circle radius equal 0, then they have to be the exact same point to have a match.

This position lookup distance is defined either by an additional event input field value, or by the parameter **`lookupdistance`**.

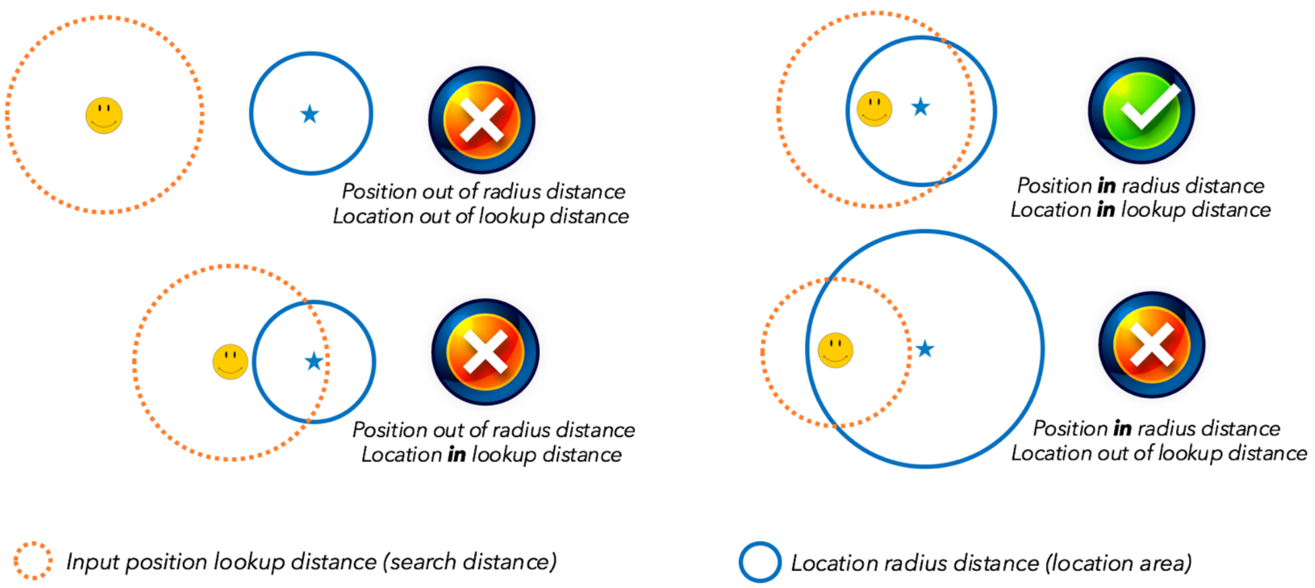
Position out of radius distance
Location out of lookup distance
Position **in** radius distance
Location **in** lookup distance
Position out of radius distance
Location **in** lookup distance
Position **in** radius distance
Location out of lookup distance
Input position lookup distance (search distance)
Location radius distance (location area)

Figure 4. Circle's Geometry Matching Logic

If there are multiple matching geometries in the lookup distance and if the option **`output-multiple-results`** is set to true, multiple events will be produced (one per geometry).

As when using polygons, the Geofence window behaves like a lookup join, so its output schema is automatically defined and includes all fields coming from the input streaming window appended with the following additional fields:

- A mandatory field of type int64 or string that will receive the ID of the geometry. If no geometries are found, the value of this field will be null in the produced event. This field is defined by the parameter **`geoid-fieldname`**.
- An optional field that will receive the description of the geometry if it exists in the geometry window schema. This field is defined by the parameter **`geodesc-fieldname`**.
- An optional field of type double that will receive the distance from the position to the center of the circle or to the centroid of the polygon. This field is defined by the parameter **`geodistance-fieldname`**.
- If **`output-multiple-results`** is set to true, a mandatory key field of type int64 that will receive the event number of the matching geometry. This field is defined by the parameter **`eventnumber-fieldname`**.

Below is a sample event stream processing XML model that implements a Geofence window using circle geometries:

```
<project name="geofencedemo" pubsub="auto" threads="4" index="pi_EMPTY">
 <contqueries>
  <contquery name="cq1" trace="alerts">
   <windows>
    <window-source name="position_in" pubsub="true" insert-only="true">
     <schema>
```

```
      <fields>
       <field name="pt_id" type="int64" key="true"/>
       <field name="GPS_longitude" type="double"/>
       <field name="GPS_latitude" type="double"/>
       <field name="speed" type="double"/>
       <field name="course" type="double"/>
       <field name="time" type="stamp"/>
       </fields>
     </schema>
    </window-source>
    <window-source name="circles_in" pubsub="true" insert-only="true">
      <schema>
       <fields>
        <field name="GEO_id" type="int64" key="true"/>
        <field name="GEO_x" type="double"/>
        <field name="GEO_y" type="double"/>
        <field name="GEO_radius" type="double"/>
        <field name="GEO_desc" type="string"/>
       </fields>
      </schema>
     </window-source>
     <window-geofence name="geofence_circle" index="pi_EMPTY">
      <geofence
            coordinate-type="geographic"
            log-invalid-geometry="false"
            output-multiple-results="true"
            output-sorted-results="true"
            max-meshcells-per-geometry="200"
            autosize-mesh="true"
            />
      <geometry
            desc-fieldname="GEO_desc"
            x-fieldname="GEO_x"
            y-fieldname="GEO_y"
            radius-fieldname="GEO_radius"
            radius="0"
            />
      <position
            x-fieldname="GPS_longitude"
            y-fieldname="GPS_latitude"
            lookupdistance="110"
            />
      <output
            geoid-fieldname="GEO_id"
            geodesc-fieldname="GEO_desc"
            eventnumber-fieldname="event_nb"
            geodistance-fieldname="GEO_dist"
            />
     </window-geofence>
    </windows>
    <edges>
     <edge source="position_in" target="geofence_circle"/>
     <edge source="circles_in"  target="geofence_circle"/>
    </edges>
   </contquery>
  </contqueries>
 </project>
```

HIGH PERFORMANCE CONSIDERATIONS

In order to provide fast and low latency lookup processing, the Geofence window is implementing an optimized mesh index algorithm using a spatial data structure that subdivides space into buckets of grid shapes called cells. This mesh structure is totally independent of the coordinate system in use. Therefore, any type of Cartesian, geographic, or projection coordinates space can be used seamlessly.

This mesh algorithm is using a parameter (called mesh factor) that defines the scale of the space subdivision. This mesh factor is an integer in the [-5, 5] range, representing a power of 10 of the coordinate units in use. For example, the default factor of 0 will generate 1 subdivision per coordinate unit, a factor of 1 generates 1 subdivision per 10 units, and a factor of -1 generates 10 subdivisions per unit. This factor can be set for both X and Y axes or independently for each axis.

For example, considering the following set of coordinates representing a square polygon (note the repeated first point at the end, closing the polygon):

```
[(1001.12,9500.12) (1001.12,9510.12) (1010.12,9510.12) (1010.2,9500.12)
(1001.12,9500.12)]
```

- With a mesh factor of 1, the Geofence window divides the coordinates by 10^1 resulting in `[(100,950) (100,951) (101,950) (101,951)]` and creates 4 mesh cells for this geometry. `(101-100+1)*(951-950+1) = 4`
- Similarly, with a factor of 2, it creates `(10-10+1)*(95-95+1) = 1` mesh cell.
- If the mesh factor is set to -1, then the window creates 9191 mesh cells for this geometry resulting in an oversized mesh: `(10101-10011+1)*(95101-95001+1)=91*101 = 9191`

As a result, in order to get the best performance, you need to adapt the mesh factor to the spatial coverage and to the number of loaded geometries. Too many mesh cells per geometry slows down the ingestion of geometries and generates an oversized index. Too few mesh cells per geometry slows down the lookup process, which impacts the stream performance and latency.

From our experience, an appropriate and efficient factor subdivides the space in order to have between 0.5 and 10 geometries per cell. This represents having each geometry generating around 1 to 10 subdivision cells maximum.

You can set the maximum allowed mesh cells created per geometry in order to avoid creating an oversized mesh that would generate useless intensive calculations using a dedicated parameter called **`max-meshcells-per-geometry`**. If a geometry exceeds this limit, it is rejected. Consider then setting a higher mesh factor or setting a higher maximum mesh cells per geometry if relevant.

The Geofence Window provides an internal algorithm that automatically computes and sets an appropriate mesh factor by analyzing the ingested geometries. If for some reason, you want to define the mesh factors manually, set the parameter **`autosize-mesh`** to false.

Using an appropriate mesh, this window has been designed to provide outstanding performance despite the number of calculations involved.

A test has been performed with a set of 21,569,300 polygons representing 625,451,932 points (~ 28 points per polygon).

With a stream of 10 million events representing 10 million different positions, the observed throughput was ~200K events/second using 1 core.

This performance level will be largely enough for most use cases, although higher performance can be easily reached by adding another window and partitioning the stream.

CONCLUSION

The new Geofence window is an easy-to-use, fast, and flexible SAS® Event Stream Processing window that provides new capabilities for processing geolocation data in real time. It allows expanding the application of streaming analytics by analyzing movements and locations of people or connected objects, and opening up the horizon to new Internet of Things applications on countless domains, in order to react immediately and appropriately.

RECOMMENDED READING

- *SAS® Event Stream Processing User Guide*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Frederic Combaneyre
SAS Institute Inc.
frederic.combaneyre@sas.com