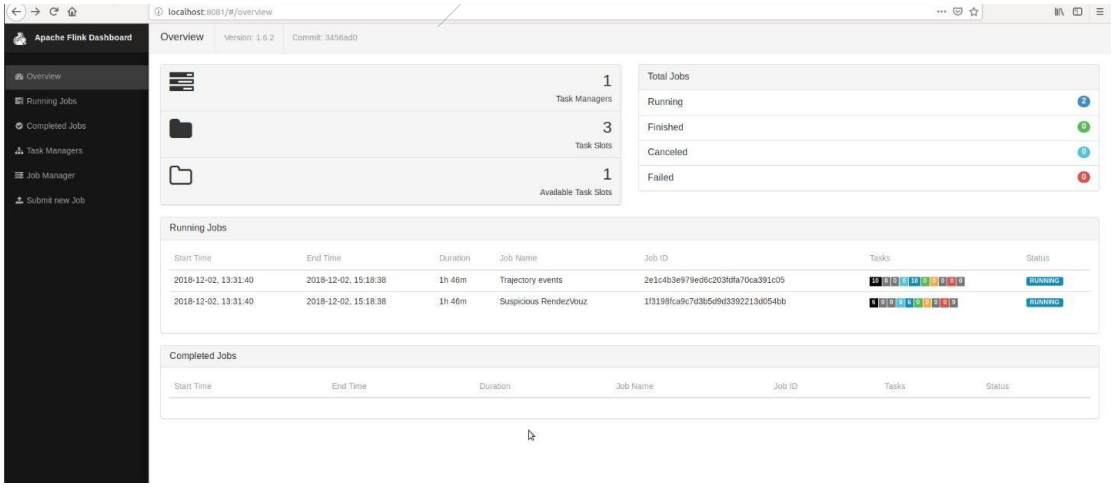
ΔΗΜΟΚΡΙΤΟΣ
ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΕΡΕΥΝΑΣ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

MSC Data Science

Applied Data Mining

Part I: Automata-based CER with FlinkCEP



Alevizopoulou Sofia 2022201704002

Avgeros Giannis 2022201704003

Tsiatsios George 2022201704024

Athens 2018

Contents

Table of Patterns

Table of Figures

Introduction

A system for online monitoring marine activity over streaming positions from numerous vessels sailing at sea will be presented in this assignment. The system detects abnormal behavior in AIS messages emitted from vessels across time. AIS messages will be used to recognize suspicious events of one or multiple vessels.

System Architecture

The online monitoring system is a combination of the following systems: kafka and Apache Flink. Kafka is an open source streaming platform which is used for stream processing and apache Flink is a real time processing engine for stateful computations.

A Flink program is defined data streams and their transformations. A stream is a flow of events, and a transformation is an operation that takes one or more streams as input, and produces one or more output streams as an output. When executed, Flink programs are mapped to directed acyclic graphs, consisting of streams and transformation operators. Each graph starts with one or more sources and ends in one or more sinks.

System Deployment

The system implemented detects complex patterns in a stream of AIS messages. AIS messages are stored in a PSQL database. A python script has been written in order to fetch these messages from database and forward them to a kafka topic named “AIS”.

The system consists of 2 different jobs. The 1st one is used to detect trajectory events for one vessel and the 2nd to detect complex event for more than one vessels based on the trajectory events that have been already detected at the previous step. The 1st project’s “flinkcep” input is what is written at topic AIS (the ais messages fetched from database). The output of this project is forwarded to other kafka topics like OUT_GAP, OUT_COTRAVEL, OUT_COURSE that given as input to the kafka producer of the 2nd project “cep_flinkcep”. The results of both projects are written in txt and csv files. Each row of the txt/csv files contains information for the detected event. Csv files will be used to make some visualizations.

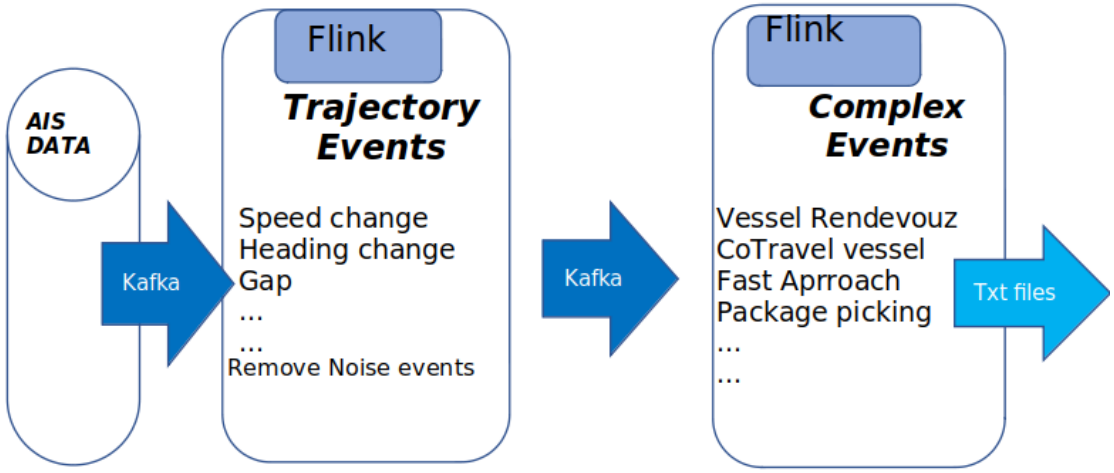


Figure 1: System deployment

Online Noise Reduction

AIS dataset, is a dataset that contains contains data which can be noisy, and as a result, difficult to be processed. Aiming to have better and more accurate results in the monitoring system, we should remove these noisy messages. In this implementation critical conditions have been added at the flinkcep patterns to identify noisy AIS messages, for examples emitted messages where the shift of the heading is more than 60 degrees. These conditions are checked firstly in the patterns in order to improve the performance of the system.

Out of order events

As it is known, transmission delays may frequently occur between the original message and its arrival. Successive positional messages from a single vessel may often arrive intermingled at a distorted order. In our case, we used custom watermarks to handle the timestamp of events and we specifically decided to not accept events arriving out of order. The main reason behind the decision was that the program's execution time was exponentially bigger if we should consider as pattern candidates the unordered events. If this system was in a real world at which we would have delays, we should take care of messages that arrived out of order by adding a small time window of accepted events despite the fact of having out of order timestamps.

Grid partitioning

We decided to translate use the coordinates of a vessel and check how far is from an area or how far is from other vessels *geohash*, a grid portioning method has been used. Geohash is a geocoding system based on a hierarchical spatial data structure which subdivides space into buckets of grid (github.com/davidmoten/geo).

Each cell is labeled using a geohash which is of user-definable precision:
- High precision geohash have a long string length and represent cells that cover only a small area.
- Low precision geohash have a short string length and represent cells that each cover a large area.

GeoHash, can have a choice of precision between 1 and 12. As a consequence of the gradual precision degradation, nearby places will often present similar prefixes. The longer a shared prefix is, the closer the two places are. In the current implementation the below cell dimensions have been used.

We preferred geohash over haversine distance because it produced must faster results for the given latitudes/longitudes.

The following table is presented as an example of the translation between geohash string length and actual distance metric units.

Precision	**Cell dimension**
4	39,1 km x 19,5 km
5	4,9 km x 4,9 km
6	1,2 km x 600,4m
7	152,9km x 152.4m

Trajectory Events

Trajectory Detection is the main module of the project since the Complex Event Recognition module uses its outcome to compute the patterns that satisfy the conditions predefined. Trajectory Movement can be vessel Speed Change Events, gap in communication and generally events that related with the movement and the behavior of one vessel or more, but mainly serve as a filter for complex events that will be checked for pattern recognition in the following project.

At this phase, system deduce instantaneous events by examining the trace of each vessel alone. This system consumes a stream of AIS tracking messages from vessels and continuously detects important patterns that characterize their movement.

A sample ais message that is consumed by the patterns written for trajectory detection is:

"lat":2.541122,"lon":3.90484,"mmsi":14,"status":7,"speed":30,"turn":,"heading":36, "course":13,1, "t": 1443650402

Trajectory detection creates a stream of important pattern of events that will be used from the complex event recognition system. Each such event is accompanied by the coordinates or some other characteristics of each corresponding vessel. At each class of trajectory events there is a message serializer class that serializes the accepted events and create another stream. These produced streams will either be used from the 2nd complex event recognition system or these results will be written on txt/csv files if there is no need to forward the stream to the 2nd monitoring system.

Below are described the trajectory events we have implemented.

1. Gaps in communication

System detects vessels which has communication gaps on sending ais messages. We can define a gap as "*the absence of emitted AIS messages from a specific vessel for a specific period*". In our approach, the ground truth of this time is 600 secs. So, if the system won't receive an ais message from a vessel between 600 secs an its far away from a port (grid 1,2 km x 600,4m), this vessel is tagged as suspicious. This pattern characterized by the critical events "gap_start" – "gap_end". Information about the geohash of ports has been fetched from database.

The flinkcep pattern is written bellow. As time window we have used 900 seconds. That all the continuous ais messages have to differ 600 secs in order to be detected from the pattern and the message gap must not exceed the value of 900.For larger windows we faced problem with the ram (heap exceeded).

Table 1: Gap Pattern

```
private static int gapTime=600;

private static int geoHashLen=6;

public static Pattern<AisMessage, ?> patternGap(){

Pattern<AisMessage, ?> rendezvouzPattern = Pattern.<AisMessage>begin("gap_start",
AfterMatchSkipStrategy.skipPastLastEvent())

  .next("gap_end")

   .where(new IterativeCondition<AisMessage>()

@Override

public boolean filter(AisMessage event,Context<AisMessage> ctx) throws Exception {  for (AisMessage ev :
ctx.getEventsForPattern("gap_start")) {

    if ((event.getT() - ev.getT()) > gapTime && (event.getT() - ev.getT()) > 0

      && listOfPorts.contains(GeoHash.encodeHash(event.getLat(),    event.getLon(), geoHashLen)) ==
false)

    {return true;

     }

  }).within(Time.seconds(1200));
```

Vessel speed according its type

In this case, the system detects vessels which have very small or very big speed for the vessel type. Each vessel type has its own min and max speeds defined, so in any case that a vessel overcomes these limits can be considered as an event worth recognizing. We obtained and store the information about the ship type of each vessel we have created a csv file which contains all vessel types and all the mmsis for each type (these data fetched from nari_static table on database). For the maximum and minimum speed of each vessel type, there is also a csv file that contains max and min speed per vessel type. This file created from information that is available on web.

The flinkcep pattern is written bellow. As time window we have 30 secs. That means at the time window of 30 secs all the ais messages comes from one mmsi have to satisfy the following conditions in order to be detected from the pattern.

Table 2: Too slow or too fast vessel pattern

```
public static Pattern<AisMessage, ?> patternSpeedVesselType(){
  Pattern<AisMessage, ?> spaciousSpeedPattern =
Pattern.<AisMessage>begin("speed_spacicious_start",

AfterMatchSkipStrategy.skipPastLastEvent())
      .followedBy("speed_spacicious_stop")
      .where(new IterativeCondition<AisMessage>() {
       public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
          for (AisMessage ev : ctx.getEventsForPattern("suspicious_heading_start")) {
                  String mmsi_= String.valueOf(event.getMmsi());
                  String type="";
                  for (Object o : listOfVesselsType.keySet()) {
                      if (Arrays.asList(listOfVesselsType.get(o)).contains(mmsi_))
                          {
                                  type = o.toString();
                          }
                  }
                  if (type.equals("")==false) {
                      String speed [] = listOfVesselsMaxMinSpeed.get(type);
                      if ( ((event.getSpeed()<Float.valueOf(speed[0]))  ||
                      (event.getSpeed()>Float.valueOf(speed[1])))   && (event.getT() - ev.getT()) > 0) {

                          return true;
                      } else {
                      ……….……
}).within(Time.seconds(30));
```

Adrift - Course of ground (COG) differentiates from the heading of a vessel

In this case, the system detects vessels with different heading and course. This can happen when there are extreme weather conditions and vessels are unable to follow the desired route. Heading describes the direction that a vessel is pointed at any time relative to the magnetic north pole or geographic north pole, of 511 degrees indicates noise. As such, a stationary vessel ex. a vessel which has been tied to a dock will have a heading associated with the vessel's orientation. COG describes the direction of motion with respect to the ground that a vessel has moved relative to the magnetic north pole or geographic north pole. An alert sign could be sent in this case. COG equals to 511 means unavailable gps, and these messages will be excluded. Accepted difference between heading and course over ground is every difference less than 10 degrees whereas suspicious can be considered every difference bigger than 10 and smaller than 60 degrees. At this case we also check the speed of a vessel, if the speed of the vessel is less than 1 KNOT it means that the vessel is anchored, and we do not accept those events. We care about vessels whose speed is between 1-48.6 KNOTS, this indicates that the vessel is under way. [1]

The flinkcep pattern is a simple condition and there is no window time.

Table 3: Heading and CoG pattern

```
static int minDiff=10;
static int maxDiff=60;
static double max_speed=48.6;
static double min_speed=1;
public static Pattern<AisMessage, ?> patternSpaciousHeading(){
 Pattern<AisMessage, ?> spaciousHeading = Pattern.<AisMessage>begin("suspicious_heading_start")
      .where(new SimpleCondition<AisMessage>() {
     @Override
      public boolean filter(AisMessage event) throws Exception {
          float courseDiffToHead=(Math.abs(event.getHeading()-event.getCourse()));
          if (courseDiffToHead>minDiff && courseDiffToHead<maxDiff && event.getSpeed()>min_speed
              && event.getSpeed()<max_speed) {            return true;      …….
```

High speed near port

At this case system detects vessels whose speed is bigger than 5KNOTS, they are near ports (grid 1,2 km x 600,4m) and at least 10 consecutive messages of a vessel satisfy those conditions. From the moment that we identify this behavior, the program does not search for intermediate patterns and follows "*skip after match strategy*". Ships that overcome these speed limits can be considered as an ongoing suspicious vessel following a dangerous route. Information about the geohash of ports has been fetched from database.

The flinkcep pattern is written bellow. As time window we have 600 secs. That means at the time window of 600 secs all the ais messages from one mmsi, have to satisfy the following conditions in order to be detected from the pattern.

Table 4:High Speed near port

```
static int indexNearPorts=6;

static int maxSpeed=5;

static int patternTime=600;

static int patternTime=600; public static Pattern<AisMessage, ?> patternSpeedNearPort(){

  Pattern<AisMessage, ?> fastForwardPattern = Pattern.<AisMessage>begin("acceleration_start",
AfterMatchSkipStrategy.skipPastLastEvent())

      .where(new SimpleCondition<AisMessage>() {

       @Override

       public boolean filter(AisMessage event) throws Exception {

          if (listOfPorts.contains(GeoHash.encodeHash(event.getLat(), event.getLon(),
indexNearPorts))== true

                    && event.getSpeed()>maxSpeed) {

                      return true;

          }return false;

             }}).times(10)

           .consecutive()

           .within(Time.seconds(patternTime));
```

Long Term Stop

Long-Term stop is only triggered if the vessel is noticed to move slowly after a pause. Pause means that vessel's speed is less than 1KNOT. [1] If the next M messages, within 200 seconds are inside a predefined area (grid 1,2 km x 600,4m) , a long term stop is identified. More specifically, this pattern first detects vessels whose speed is smaller than 1KNOT and far away from ports (grid 1,2 km x 600,4m). After check if the detected vessels don't change their speed more than 2 KNOTS within 1800 secs and they send ais messages inside an area (grid 1,2 km x 600,4m), means they haven't moved away. Information about the geohash of ports has been fetched from database.

The flinkcep pattern is written bellow. As time window we have 7200 secs. That means at the time window of 7200 secs all the ais messages from one mmsi have to satisfy the following conditions in order to be detected from the pattern.

Table 5: Long Term Stop

```
public static Pattern<AisMessage, ?> patternLongStop() {

    int longterm = 3600;

    int time_window=7200;

    Pattern<AisMessage, ?> LongTermStopPattern = Pattern.<AisMessage>begin("start",
AfterMatchSkipStrategy.skipPastLastEvent())

        .next("stop")

        .where(new SimpleCondition<AisMessage>() {

          @Override

          public boolean filter(AisMessage event) throws Exception {

            if((event.getSpeed() <1 && listOfPorts.contains(GeoHash.encodeHash(event.getLat(),
event.getLon(), 8)) == false)){

              String geoHash1=GeoHash.encodeHash(event.getLat(),event.getLon(),8);

              return true;

            }

            return false;

          }

        })
```

```
            .followedBy("stop_ends")
            .where(new IterativeCondition<AisMessage>() {
               @Override
               public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
                  for (AisMessage ev : ctx.getEventsForPattern("stop")) {
                     if ( ev.getMmsi() == event.getMmsi()) {
                        String geoHash1=GeoHash.encodeHash(event.getLat(),event.getLon(),6);
                        String geoHash2=GeoHash.encodeHash(ev.getLat(),ev.getLon(),6);
                        if((geoHash1.equals(geoHash2)) && (event.getSpeed()<2) && ev.getSpeed()<2  &&
(Math.abs(event.getT() - ev.getT()) > 1800)){
                           return true;
                        }
                        else{
                           return false;
                        }
                     }.within(Time.seconds(time_window));
                      …..
                     return false;    }})
```

Vessels with false status

The dataset contains vessels with wrong status for example vessels with speed more than 5KNOTS and status moored or at anchor (status =1, status=5). Those ships could be identified as noise.

The flinkcep pattern is a simple condition and there is no window time.

Table 6: Vessels with false status

```
public static Pattern<AisMessage, ?> patternFalseType() {

    Pattern<AisMessage, AisMessage> alarmPattern =
Pattern.<AisMessage>begin("first")

        .where(new SimpleCondition<AisMessage>() {

        @Override

        public boolean filter(AisMessage event) {

            if ((event.getStatus() == 1 || event.getStatus() == 5)) {

              if (event.getSpeed() > 5) {

                 return true;

              } else

                 return false;

             ……..

        });
```

Complex event

Complex event Recognition module consumes the output of the Trajectory detection module process the results and recognize in real time potentially complex maritime situations. Below will be described some patterns used for complex event recognition. For some of them, two monitoring systems needed. The 1st system for the trajectory events that are analyzed at the previous section and the 2nd one for combining these events to detect more complex patterns.

1. Two vessels co-travelling

The pattern, checks that the ais messages from 2 different vessels are between a time period of 15 secs. The pattern will detect events that happened closely in the time dimension. Next thing that will be checked is if the ships are on route and not paused. The speed of the vessel should be bigger than 1KNOT (trajectory event - 1 KNOT is the minimum speed of a vessel that isn't in anchor). [1] At the end, the geohash of the 2 vessels will be checked, ensuring by that, that the two vessels were located in the same grid. The precision of geohash is 6 (grid: 1.2 km x 600m). Two vessels seem to cotravel if they are close enough (grid: 1.2 km x 600m) for more than 180 secs. This pattern characterized from events "vessel_1" – "vessel_2".

At the end,all the detected patterns will be inserted into a kafka topic and will be given as input stream at the 2nd online monitoring system "cep_flincep" to detect the desired events.

In order to make the second job more efficient, we decided to insert the id of the vessel that is the smallest between mmsi 1 and 2 as MMSI_1 and to collect all the events per vessel. With that, instead of search through all the accepted events we will consider only the pairs that at least one of the MMSI does not change.

As time window we have used 15 secs for the 1st pattern and 300 secs for the 2nd. 15 secs of the 1st pattern means all the ais messages (for all vessels) in a time window of 15secs should be at the same geohash in order to be detected from the system. The second pattern checks that vessels emitted from the first pattern, follow the following rules:

- Two different messages for two different ships that co-exist in the same grid must at worst be differentiated by a margin of 35 seconds.
- Going backwards to the accepted messages following the first rule, we check whether those messages surpass in total the 180 seconds margin. Base event is considered the latest, and we move iteratively backwards until we find a series of messages that the difference between an event and the base event is more than 180 seconds.

- All this should happen between 300 seconds.

The 1st pattern is :

Table 7: Cotravel for 2 vessels pattern – trajectory events

```
static int minSpeed=1;
static int geoHashLength=6;
static int timeBetweenVesselsMsgs=15;
public static Pattern<AisMessage, ?> patternCoTravel(){
  Pattern<AisMessage, ?> coTravelPattern = Pattern.<AisMessage>begin("vessel_1")
 .subtype(AisMessage.class)
 .followedBy("vessel_2")
 .subtype(AisMessage.class)
 .where(new IterativeCondition<AisMessage>() {
@Override
public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
for (AisMessage ev : ctx.getEventsForPattern("vessel_1")) {
   if(ev.getSpeed()>minSpeed
    && event.getSpeed()>minSpeed
    && ev.getMmsi()!=event.getMmsi()){
     String geoHash1=
     GeoHash.encodeHash(ev.getLat(),ev.getLon(),geoHashLength);
     String geoHash2=
 GeoHash.encodeHash(event.getLat(),event.getLon(),geoHashLength);
        if(geoHash1.equals(geoHash2)==true){
          return true;
         }else{return false;
          }……………
           .within(Time.seconds(timeBetweenVesselsMsgs));
```

The 2nd pattern is:

Table 8: Cotravel for 2 vessels pattern – complex events

```
static int coTravelTime=35;

static int coTravellingTotalTime=180;

Pattern<CoTravelInfo, ?> coTravelattern =
Pattern.<CoTravelInfo>begin("msg_1",AfterMatchSkipStrategy.skipPastLastEvent())

  .subtype(CoTravelInfo.class)

   .oneOrMore()

   .followedBy("msg_2")

   .where(new IterativeCondition<CoTravelInfo>() {

   @Override

  public boolean filter(CoTravelInfo event, Context<CoTravelInfo> ctx) throws Exception{

    int base = event.getTimestamp();

    int currTime = event.getTimestamp();

   List<CoTravelInfo> l = Lists.newArrayList(ctx.getEventsForPattern("msg_1"));

    for (CoTravelInfo ev : Lists.reverse(l)) {

      if ((currTime - ev.getTimestamp()) < coTravelTime) {

        if (event.getMmsi_2() == ev.getMmsi_2()) {

          if ((base - ev.getTimestamp()) > coTravellingTotalTime) {

             return true;

           } else {

               currTime = ev.getTimestamp();

           }}} else {

            return false;

      }……..

            .within(Time.seconds(600));
```

2. Fishing Activity

This pattern combines two trajectory events in order to detect a complex event. Specifically, it checks the sequence of continuous changes of the heading (changes between 15 and 60 degrees) , followed by a gap in communication (600 secs) and a final turn in the end (change between 15 and 60 degrees). This can be considered as an alert sign for illegal fishing. The captain checks the area and afterwards closes its GPS. At that point the system will check just the events that have been characterized by the tag "gap_start" – "gap_end" in order to detect some more changes in heading of the vessels. The trajectory events of heading change and gap in communication have been already analyzed.

The flinkcep pattern is written bellow. As time window we have used 1200 secs. That means at the time window of 1200 secs the ais messages from one mmsi have to satisfy the following conditions in order to be detected from the pattern.

Table 9: Fishing pattern

```
static int headingChangeMin=15;
static int headingChangeMax=60;
static int gapTime=600;
 public static  Pattern<AisMessage, ?> patternFishing(){
  Pattern<AisMessage, ?> fishingPattern = Pattern.<AisMessage>begin("start")
.subtype(AisMessage.class)
.followedBy("gap_start")
.subtype(AisMessage.class)
.where(new IterativeCondition<AisMessage>() {
 @Override
 public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
   for (AisMessage ev : ctx.getEventsForPattern("start")) {
    if(Math.abs(ev.getHeading()-vent.getHeading())>headingChange)
    {return true;
     }else{
        return false;…… }})
```

```
.subtype(AisMessage.class)
 .followedBy("gap_end")
 .subtype(AisMessage.class)
.where(new IterativeCondition<AisMessage>() {
  @Override
  public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
     for (AisMessage ev : ctx.getEventsForPattern("gap_start")) {
         if((event.getT()-ev.getT())>gapTime){
             return true;
          }else{
             return false; }}
     return false;}})
 .followedBy("change in heading  again")
 .subtype(AisMessage.class)
 .where(new IterativeCondition<AisMessage>() {
 @Override
 public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
    for (AisMessage ev : ctx.getEventsForPattern("gap_end")) {
if(Math.abs(ev.getHeading()-event.getHeading())>headingChange){
        return true;
 }else{
       return false; }}
  return false;}})
           .within(Time.seconds(600));
```

3. Vessel Rendezvous

The pattern, checks that the ais messages from 2 different vessels if they have gap in their communication are in the same geohash grid. The precision of geohash is 6 (grid: 1,2 km x 600,4m). The information about the geohash of each vessel is given by the previous pattern “gap-communication” whose outcome is used as input stream ain this pattern. Two vessels may have arranged a rendezvous while on the sea, when they have both gap in their communication at the same time as long as they are at the same geohash grid.
This pattern characterized from events “vessel_1” – “vessel_2”.

The flinkcep patterns are written bellow. As time window we have used 900 secs for the 1st pattern and 120 secs for the 2nd. 900 secs of the 1st pattern means all the continuous ais messages of a mmsi have to differ 600 secs in order to be detected. With the second pattern, we aim to identify vessels that their gap end event differs at most to 120 seconds.

The 1st pattern is:

Table 10: Vessel Rendezvous pattern - trajectory events

```
private static int gapTime=600;

private static int geoHashLen=6;

public static Pattern<AisMessage, ?> patternGap(){

Pattern<AisMessage, ?> rendezvouzPattern = Pattern.<AisMessage>begin("gap_start",
AfterMatchSkipStrategy.skipPastLastEvent())

  .followedBy("gap_end")

   .where(new IterativeCondition<AisMessage>()

@Override

public boolean filter(AisMessage event,Context<AisMessage> ctx) throws Exception {  for
(AisMessage ev : ctx.getEventsForPattern("gap_start")) {

    if ((event.getT() - ev.getT()) > gapTime && (event.getT() - ev.getT()) > 0

      && listOfPorts.contains(GeoHash.encodeHash(event.getLat(),    event.getLon(), geoHashLen)) ==
false)

    {

      return true;

     }

       ……….

  }).within(Time.seconds(900));
```

The 2nd pattern is:

Table 11: Vessel Rendezvous pattern - complex events

```
public static int gapTime=120;
public static Pattern<GapMessage, ?> patternRendezvouz(){
    Pattern<GapMessage, ?> rendevouzPattern =
Pattern.<GapMessage>begin("Vessel_1")
  .subtype(GapMessage.class)
  .followedBy("Vessel_2")
  .subtype(GapMessage.class)
  .where(new IterativeCondition<GapMessage>() {
  @Override
  public boolean filter(GapMessage event, Context<GapMessage> ctx) throws Exception {
   for (GapMessage ev : ctx.getEventsForPattern("Vessel_1")) {
     if ((ev.getGeoHash().equals(event.getGeoHash()) == true)
       && ev.getMmsi()!=event.getMmsi()) {
        return true;
     } else { return false;
     }}
   return false;}})
          .within(Time.seconds(gapTime));
```

4. Adrift

The pattern, detects vessels that may have a problematic route. The vessels checked at this pattern are already detected as vessels with suspicious difference between heading and course of ground.
As far as the second pattern is concerned, the following prerequisites must be met:

- Two consecutive messages for a vessel must be on a time range of 20 seconds.
- Going backwards to the accepted messages following the first rule, we check whether those messages surpass in total the 180 seconds margin. Base event is considered the latest, and we move iteratively backwards until we find a series of messages that the difference between an event and the base event is more than 180 seconds.

The flinkcep patterns are written bellow. The second pattern checks that vessels emitted from the first pattern, follow the following rules:

- All vessels must have minimum
- Going backwards to the accepted messages following the first rule, we check whether those messages surpass in total the 120 seconds margin. Base event is considered the latest, and we move iteratively backwards until we find a series of messages that the difference between an event and the base event is more than 120 seconds.
- All this should happen between 300 seconds.

The 1st pattern is:

Table 12: Big difference between heading and course – trajectory event

```
static int minDiff=10;
static int maxDiff=60;
static double max_speed=48.6;
static double min_speed=1;
public static Pattern<AisMessage, ?> patternSpaciousHeading(){
 Pattern<AisMessage, ?> spaciousHeading = Pattern.<AisMessage>begin("suspicious_heading_start")
      .where(new SimpleCondition<AisMessage>() {
     @Override
      public boolean filter(AisMessage event) throws Exception {
          float courseDiffToHead=(Math.abs(event.getHeading()-event.getCourse()));
          if (courseDiffToHead>minDiff && courseDiffToHead<maxDiff &&
event.getSpeed()>min_speed
                && event.getSpeed()<max_speed) {
                     return true;
          } else {
                    return false; …….
```

The 2nd pattern is:

Table 13: Problematic route - complex events

```
static int messagesTimeGap=20;
 static int problematicRouteTime=180;
   public static Pattern<courseHead, ?> patternSpaciousHeading(){
      Pattern<courseHead, ?> spaciousHeading =
Pattern.<courseHead>begin("suspicious_heading_start", AfterMatchSkipStrategy.skipPastLastEvent())
           .oneOrMore()
           .followedBy("suspicious_heading_stop")
           .where(new IterativeCondition<courseHead>() {
              @Override
              public boolean filter(courseHead event, Context<courseHead> ctx) throws Exception {
                 int base = event.getTimestamp();
                 int currTime = event.getTimestamp();
                 List<courseHead> l =
Lists.newArrayList(ctx.getEventsForPattern("suspicious_heading_start"));
                 for (courseHead ev : Lists.reverse(l)) {
                    if ((currTime - ev.getTimestamp()) < messagesTimeGap) {
                       if ((base - ev.getTimestamp()) > problematicRouteTime) {
                          return true;
                       } else {
                          currTime = ev.getTimestamp();
                       }} else {
                       return false; ……..
}).within(Time.seconds(300));
```

5. Package Picking

This pattern detects possible interaction between two vessels. A possible interaction is when one of them drops a package at some area and another vessel appears later in order to pick it up. By joining the previous long stop events using geohash area (grid: 1,2 km x 600,4m) we find ships that have long stops in the same area where the package picking is possible. As described before the ships should be away from ports. The timestamp between the vessels to be in the same geohash area is 60secs.

The flinkcep patterns are written bellow. As time window we have used 3600 secs. That means at the time window of 3600 secs the ais messages satisfy the following conditions in order to be detected from the pattern.

```
Pattern<SuspiciousLongStop, ?> alarmPattern = Pattern.<SuspiciousLongStop>begin("first")
    .where(new SimpleCondition<SuspiciousLongStop>() {
    @Override
   public boolean filter(SuspiciousLongStop suspiciousLongStop) throws Exception {
       if(suspiciousLongStop.getMmsi()>0) {
            return true; }else{
          return false; }}})
    .followedBy("picking")
    .where(new IterativeCondition<SuspiciousLongStop>() {
    @Override
   public boolean filter(SuspiciousLongStop event, Context<SuspiciousLongStop> ctx) throws Exception {
       String geoHash1= GeoHash.encodeHash(event.getLat(),event.getLon(),6);
              for (SuspiciousLongStop ev : ctx.getEventsForPattern("first")) {
                 if ( ev.getMmsi() != event.getMmsi()) {
                 String geoHash2=GeoHash.encodeHash(ev.getLat(),ev.getLon(),6);
                    if((geoHash1.equals(geoHash2)) && (event.getGapEnd() - ev.getGapEnd())<60){
                       return true; }
                   else{return false; }}) ……..
        .within(Time.seconds(3600));
```

Loitering

Loitering is the act of remaining in a particular area for a long period without purpose. Vessels with low speed, anchored or moored must be filtered out. If the messages from a single vessel are in the same area for 1800 secs (loitering time , Ltrtime) this vessel is considered to be loitering. So firstly detects vessels that are far away from ports (grid: 1,2 km x 600,4m) with speed bigger than 2.87 KNOTS and smaller than 8 KNOTS and after check id this vessel remain at the same area. Information about the geohash of ports has been fetched from database. [1]

The flinkcep patterns are written bellow. No pattern window is applied we simply search for vessels for loitering. Optional time window would be 3600 (1 hour)

```
Pattern<AisMessage, ?> Loitering = Pattern.<AisMessage>begin("stop")
    .subtype(AisMessage.class)
    .where(new SimpleCondition<AisMessage>() {
      @Override
      public boolean filter(AisMessage event) throws Exception {
        boolean near_ports = false;
        for(String str: Ports) {
          String ship_geohash = GeoHash.encodeHash(event.getLat(),event.getLon(),6);
          if(str.equals(ship_geohash))
            near_ports = true;
        }
        if((event.getSpeed() >2.87 && near_ports == false)){
          if(event.getSpeed() < 8){
            String geohash1=GeoHash.encodeHash(event.getLat(),event.getLon(),6);
            return true;}
          else{
              return false;
          ……..……. })
    .followedByAny("stop_ends")
    .where(new IterativeCondition<AisMessage>() {
      @Override
      public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {
        for (AisMessage ev : ctx.getEventsForPattern("stop")) {
          String geoHash1=GeoHash.encodeHash(event.getLat(),event.getLon(),6)
          if ( ev.getMmsi() == event.getMmsi()) {
            String geoHash2=GeoHash.encodeHash(ev.getLat(),ev.getLon(),6);
            if((geoHash1.equals(geoHash2)) && ev.getSpeed()< 8 && (event.getSpeed()< 8 && (event.getT()-
ev.getT()>ltrtime))){
              if(event.getSpeed()>2.87 && ev.getSpeed() > 2.87){
```

```
                    return true;
                }
                else{
                    return false;
                }
            }
            else{
                return false;
……….
        }});
```

Producer of ais messages for the kafka topic

The ais messages are fetched from database with a pythin script and after given as input at a kafka topic. The python script is insie the path flinkcep/producer and the script is :ais.py. Runnign the script t kafka topic anme should be defined.

Watermark Pattern

A time pattern has been used in messagestream for using as time the timestamps from ais messages and not the real time of machine.

Table 14: Watermark pattern

```
DataStream<AisMessage> messageStream = env .addSource(new
FlinkKafkaConsumer09<>(
        parameterTool.getRequired("INPUT"),new AisMessageDeserializer(),props))
        .assignTimestampsAndWatermarks(new Watermarks());
```

Running Apache Flink Jobs

As analyzed before system consists of 2 projects. For each project there is a different job running on flinkcep. For each job there are different patterns that are checked. The following screenshots show a sample execution of them. The system is running for 1h 48 min, that means system receives ais messages for 1h 48 min. During this period of time, system detects trajectory and complex events.

You can check how long the job runs as long as the number of events which has been detected.

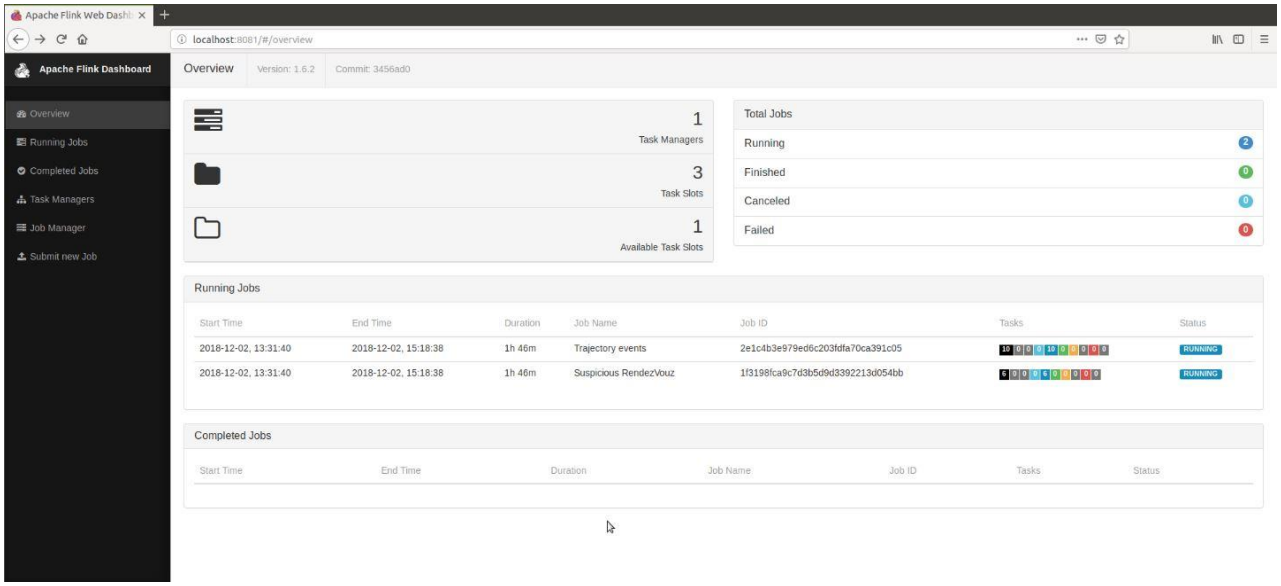


Figure 2:Two jobs are running: trajectory and complex events

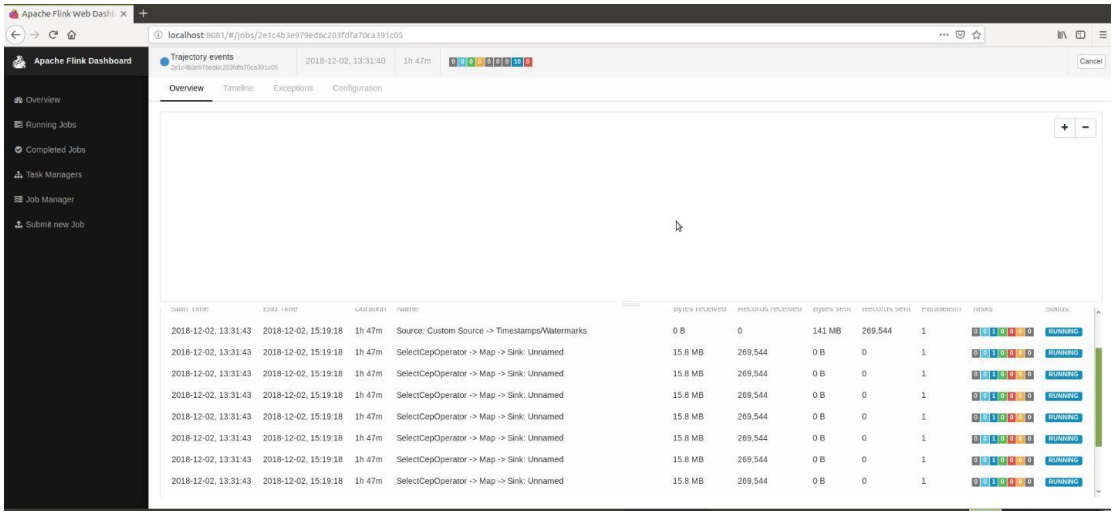


Figure 3: Patterns for jobId: trajectory event (part a)

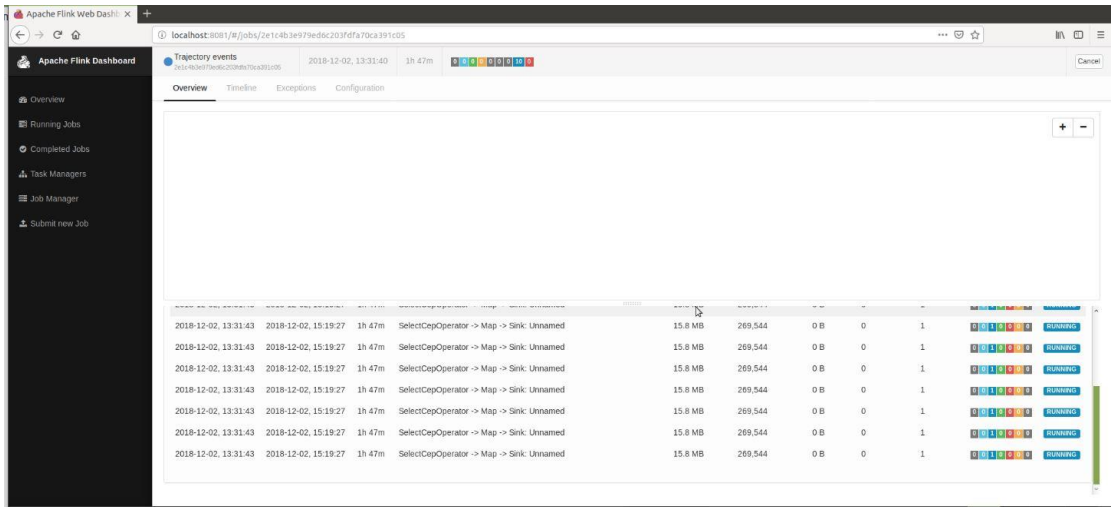


Figure 4:Patterns for jobId: trajectory event (part b)

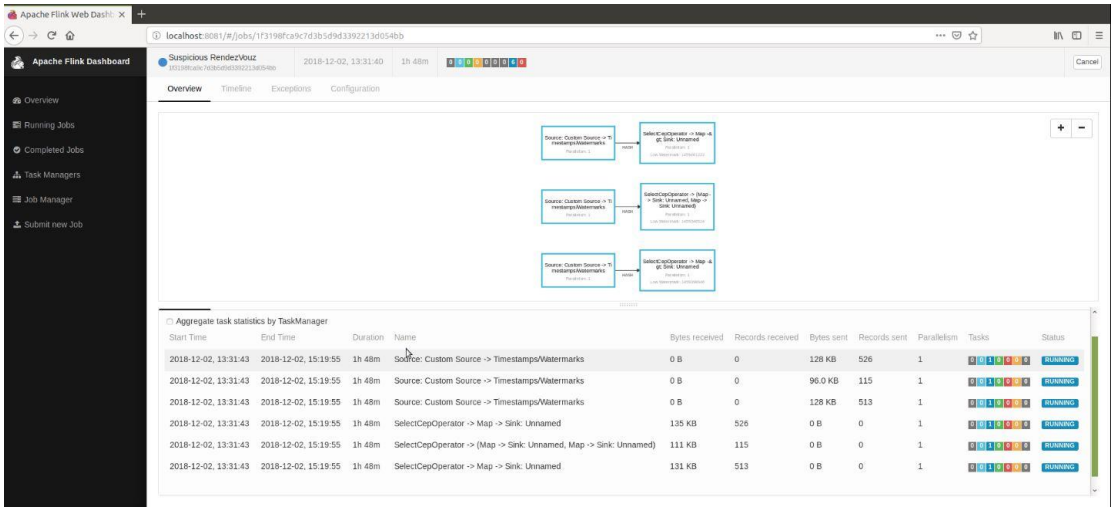


Figure 5: Patterns for jobId: complex event

At this job, which takes as input the output of the 1st job the acyclic graphs are shown, consisting of streams and transformation operators. Each graph starts with one source and end in one sink.

Outcomes of the running jobs

All the above patterns, while running they write their results (information about the detected events) into .txt files at the path /Desktop/temp. After that, some python scripts will be executed (at the path /conver_txt_to_csv) for parsing these .txt files and make .csv files which contain just the values of fields that are needed to make visualizations on qgis.

The .txt files are listed below:

- speed_near_port.txt, fishing.txt, false_speed.txt, loitering.txt, longstop.txt, packages.txt, rendezvous.txt, cotravel.txt, coursehead.txt, suspicious_speed.txt

The scripts used for these files are:

files	script
coursehead.txt	decode.py
cotravel.txt	decode5.py
loitering.txt	decode1.py
Packages.txt	decode2.py
rendezvous.txt	decode3.py
speed_near_port.txt	decode6.py
f ishing.txt	decode4.py

Empirical Evaluation

Changing the values on mobility tracking parameters eg: secs of gap in communication, secs of gap in a suspected fishing etc we can see that the number of events that are detected changes dramatically. Doing some tests with 1.000.000 ais messages, we conclude at the below results. We decided to use as parameters values these that were giving us more realistic results (the bold values).

Rendezvous

GeoHash	Detected events
5 *4.9km x 4.9km*	476
6 *1.2km x 609.4m*	432
7 ***152m x 152m***	**195**
8 *38.2m x 19m*	58

Illegal Fishing

Secs of gap in a suspected fishing activity(max 1200)	Detected illegal fishing events
300	16
600	**10**
900	9
1200	8

Speed near port

GeoHash for near port	Detected events
5 *4.9km x 4.9km*	1545
6 ***1.2km x 609.4m***	**41**
7 *152m x 152m*	0
8 *38.2m x 19m*	0

Visualization of the detected events

The below pictures come from QGIS framework using the .csv files with the several complex or trajectory events. These visualizations represent the detected patterns using the bold empirical values on patterns (see previous Section). These are also the values that are written at the patterns' description on this report.

Cotraveling activity

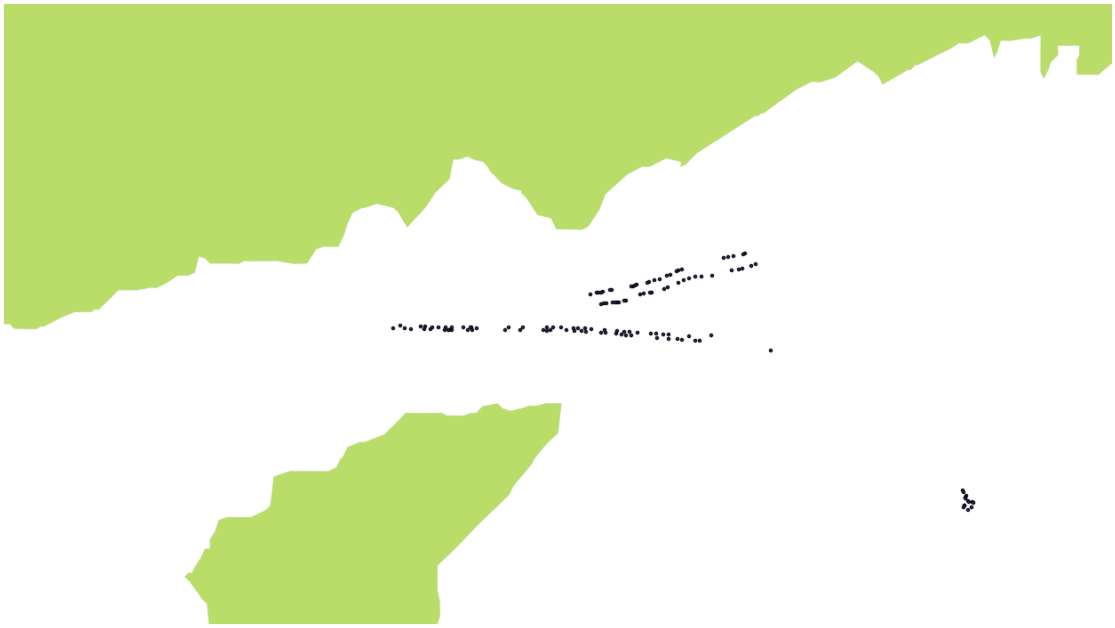


Figure 6: Cotraveling activity

Loitering activity

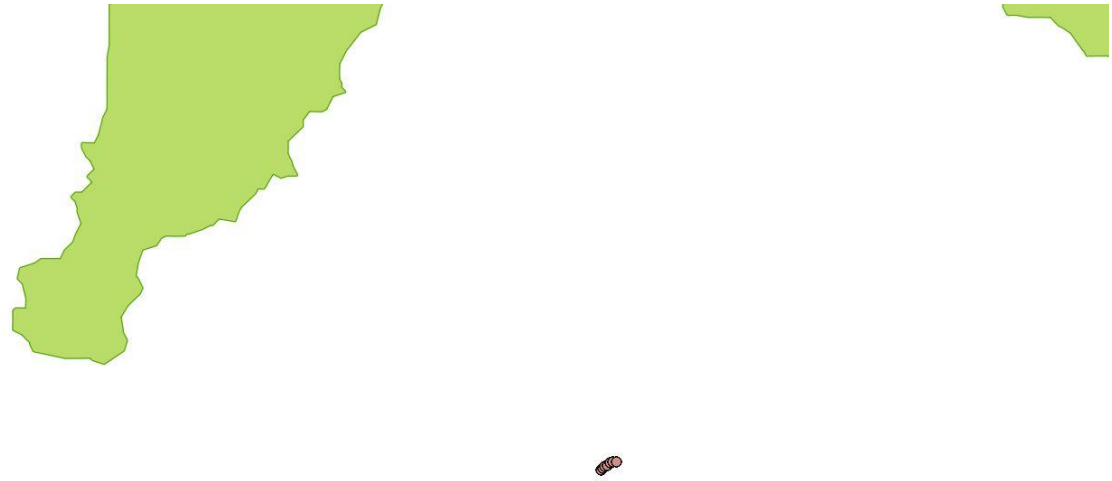


Figure 7: Loitering activity

Adrift activity

Figure 8: Adrift activity

High speed near port

Figure 9: High speed near port, grid of 1.2km x 609.4m

Figure 10: High speed near port, grid of 4.9km x 4.9km (1545 detected events)

Figure 11: Fishing activity, 300 secs gap in communication (16 detected events)

Figure 12: Fishing activity, 600 secs gap in communication (10 detected events)

Figure 13: Fishing activity, 900 secs gap in communication (9 detected events)

Figure 14: Fishing activity, 1200 secs gap in communication (8 detected events)

Vessels rendezvous

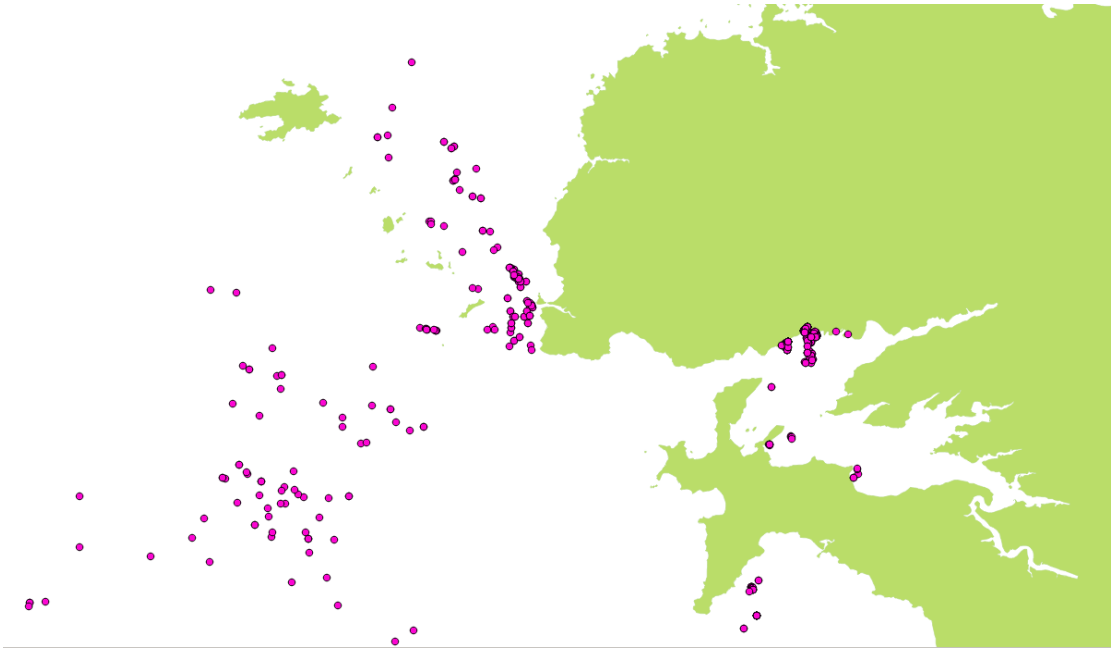


Figure 15: Vessels Rendezvous, grid of 4.9km x 4.9km (476 detected events)

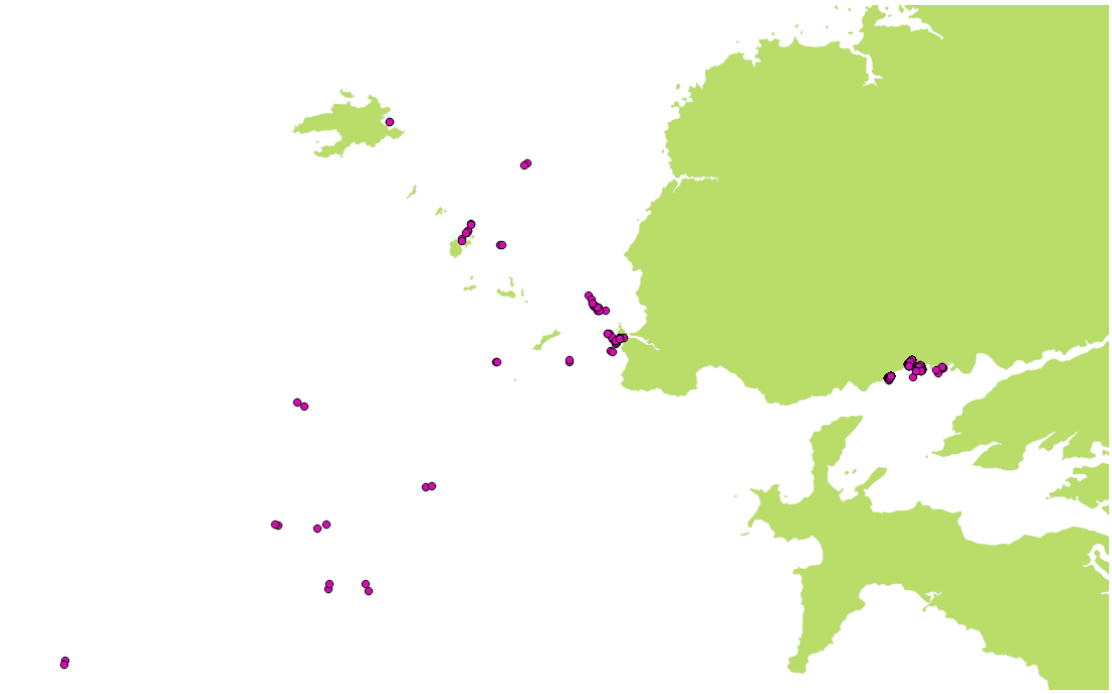


Figure 16: Vessels Rendezvous, grid of 1.2km x 609.4m (432 detected events)

Figure 17: Vessels Rendezvous, grid of 152m x 152m (195 detected events)

Figure 18: Vessels Rendezvous, grid of 38.2m x 19m (58 detected events)

Package picking

Figure 19: Package picking

Running commands

First you need to run 3 servers: flink cluster server, zookeeper and kafka server. The next step is to create 4 topics on kafka. Also, there are 2 java projects that will run and one python script which will be used as producer for the main topic of the system (the one contains ais messages).

The commands you need to execute are described below:

- ***$ flink_1.6.2:bin/start-cluster.sh***
  - start the cluster of flink
  - Check that server is running on http://localhost:8081/#/overview
- ***$ kafka2.2:bin/zookeeper-server-start.sh config/zookeeper.properties***
- ***$ kafka2.2:bin/kafka-server-start.sh config/server.properties***
  - Start the kafka and zookeeper servers
- ***$ kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 partitions 1 --topic AIS***
- ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic OUT_GAP***
- ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic OUT_COTRAVEL***
- ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic OUT_COURSE***
  - Create the topics on kafka
- ***sudo /home/cer/Desktop/flink-1.6.2/bin/flink run /home/cer/Desktop/cer/flinkcep/cep_flinkcep/target/flinkicu_cep-1.0-jar-with-dependencies.jar --IN_GAP GAP --bootstrap.servers localhost:9092 --zookeeper.connect localhost:2181 --IN_COTRAVEL COTRAVEL --IN_COURSE COURSE***
- ***sudo /home/cer/Desktop/flink-1.6.2/bin/flink run /home/cer/Desktop/cer/flinkcep/flinkcep/target/flinkicu-1.0-jar-with-dependencies.jar calhost:9092 --zookeeper.connect localhost:2181 --OUT_GAP GAP --OUT_COTRAVEL COTRAVEL --OUT_COURSE COURSE***
- ***$ Kafka2.2:bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 topic AIS***
  - You can check the events that detected at each job at this url. http://localhost:8081/#/overview
  - You can see what is send at each topic running the consumer of each topic. For example for topic AIS, running the consumer you will see the ais messages
- ***./ais.py AIS***

- Run the python script in order to full fill the topic that contains all the ais messages and 1st module starts receiving ais messages
- Producer is inside the project at the path ‘flinkcep/producer’

Running the script suspicious events will start to be detected. All of these will be written at the path *home/cer/Desktop/temp/* as .txt /.csv files. There will be a .txt /.csv file for each trajectory and suspicious event.

Repository

The source code is located at:

https://github.com/salevizo/flinkcep.git

Source code

- 2 java maven projects: implement flink patterns for detecting trajectory or complex events
    - *Flinkcep*
    - *Cep_flinkcep*
- 1 python script *ais.py* inside flinkcep that send ais messages inside kafka producer
- 7 python scripts to convert the outcomes of the projects .txt files to csv in order to plot them on qgis, located inside folder */convert_txt_to_csv*.
- 1 folder */outcomes* which contains all the generated outcomes of 4 different executions of the project.
- Running the project, the outcomes will be saved at the path /home/cer /Desktop/temp
- README file
- The project is located at the path:
    - /home/cer/Desktop/cer_2/flinkcep/
- Kafka is located at the path:
    - /home/cer/Desktop/
- Flink is located at the path:
    - /home/cer/Desktop/

Running environment

OS	PRETTY_NAME="Ubuntu 18.04.1 LTS" VERSION_ID="18.04"
Postgres	postgres=# SELECT version(); PostgreSQL 10.5 (Ubuntu 10.5-0ubuntu0.18.04) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 7.3.0-16ubuntu3) 7.3.0, 64-bit
QGIS	2.18.0
Python	Python 2.7.15
Kafka	kafka_2.11-1.0.0
Flink	flink-1.6.2
Scala	Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/E

References

[1] Online Event Recognition from Moving Vessel Trajectories Kostas Patroumpas · Elias Alevizos · Alexander Artikis · Marios Vodas · Nikos Pelekis · Yannis Theodoridis.

[2] COMPARISON OF INS HEADING AND GPS COG, R. Michael Reynolds November 6, 20

http://www.rmrco.com/docs/m1227_Compare-cog-hdg.pdf