7 July 2021

Université
Paris-Dauphine

Data Streaming

Report for the Ad fraud detector project

Yves Tran
Alban Tiacoh

Contents

Abstract

This report explains how we implemented our Flink application in order to filter suspicious activities from a Kafka producer of an online advertiser through. It sums up the analysis we did to identify fraudulent patterns. Two outputs are produced, a first stream sinks suspicious events to an output file and a second one containing events with further information extracted from the implementation of rule sinks data to an ElasticSearch-Kibana stack in order to monitor activities and the CTR through time.

1 Analysis

This section sums up the offline analysis to detect suspicious patterns. For a more complete view and understanding, one can look at the IPython notebook named *analysis/Analysis.ipynb* from the code repository.

1.1 Hypothesis

In a pay-per-click context, we suppose the main motivation of fraudulent clicks is to exhaust the advertiser budget. This can be done by hiring users to click on ads or creating bots tasked to click on any ads. Thus, we can formulate some hypothesis about the behaviour of scammers based on the number of click they generate and their time reaction such as :

- bots might click instantly,
- abusive clicks,
- a recurrent user or IP address,
- a very short reaction time from an user.

1.2 Methodology

In order to check these hypothesis and validate/invalidate the suspicious patterns , we have recorded 2 hours and a half of activities as a train set to discover suspicious patterns and another 1-hour-and-a-half sequence of activities to create a test set. Each pattern has been discovered once previous patterns have been removed so that there is no interference. We will use the CTR (Click-Through-Rate) defined by the number of clicks divided by the number of displays in a specific interval of time. In order to retain enough information between activities, we consider time windows of 10 minutes. A normal CTR should be around 10%.

1.3 Gathering data

Originally, events were sinked to ElasticSearch and primary analysis could be done thanks to Kibana.

For more fine-grained analysis, we end up using Jupyter notebook to visualise data. The train and test sets have been created using the *logger* package we implemented. It reads the output events from Kafka and write them to a file to be analysised.

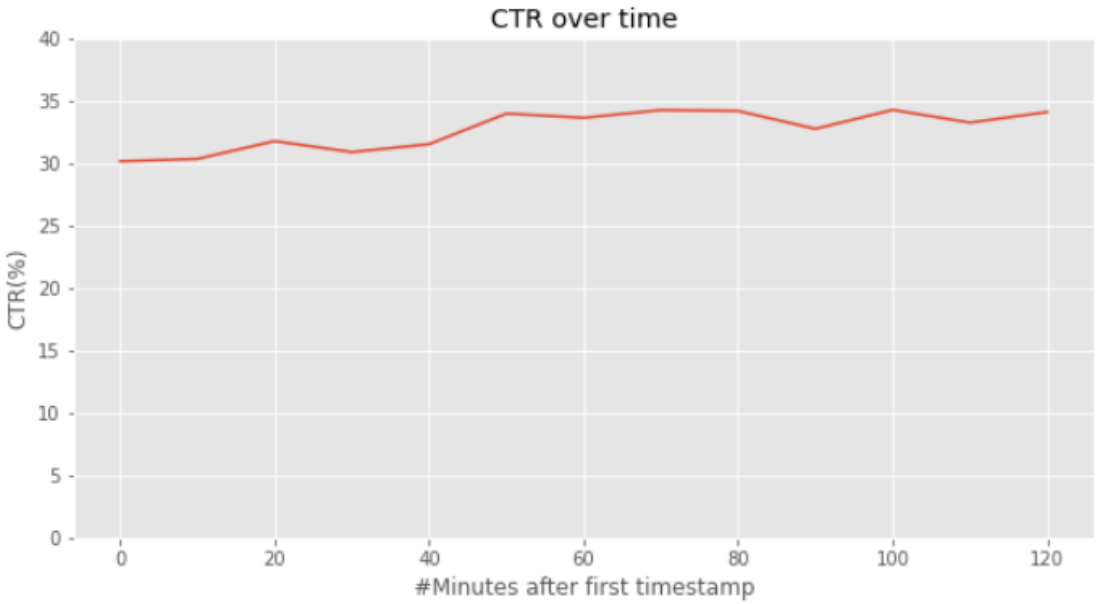

Figure 1: The evolution of the CTR is computed within a time window of 10 minutes from the training set. We can see the CTR is around 30% at each period.

1.4 Suspicious patterns

1.4.1 First pattern

From the train set, we have discovered a recurrent IP address that is responsible of 1200 events (600 clicks and 600 displays) every 10 minutes as shown in Fig 2 which makes our first suspicious pattern : removing activities associated with an IP address with more than 10 events.

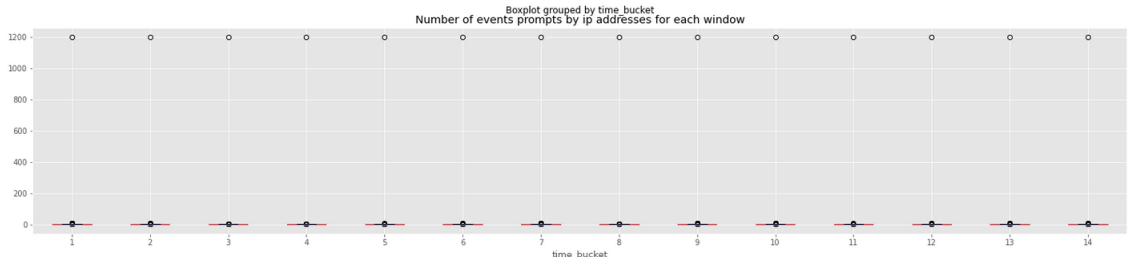

Figure 2: These boxplots show a single outlier within each time window.

1.4.2 Second pattern

To measure the reaction time of users, we computed the average duration elapsed between clicks and their corresponding displays (once the first pattern removed). Fig 3 shows users segmented by their average reaction time. We can see some users have a short average reaction time of 2-3 seconds. Thus, we consider a second pattern made of users with an average time reaction less or equal to 3 seconds within a window.

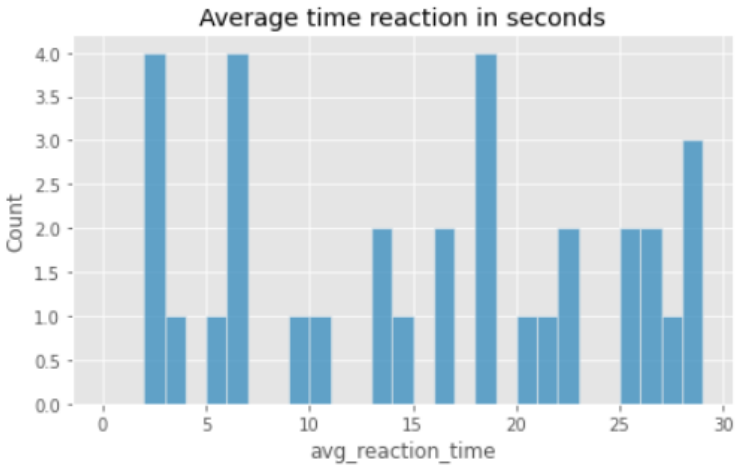

Figure 3:

1.4.3 Third pattern

We discovered a portion of users who clicked more than 20 times within 10 minutes as shown in Fig 4.

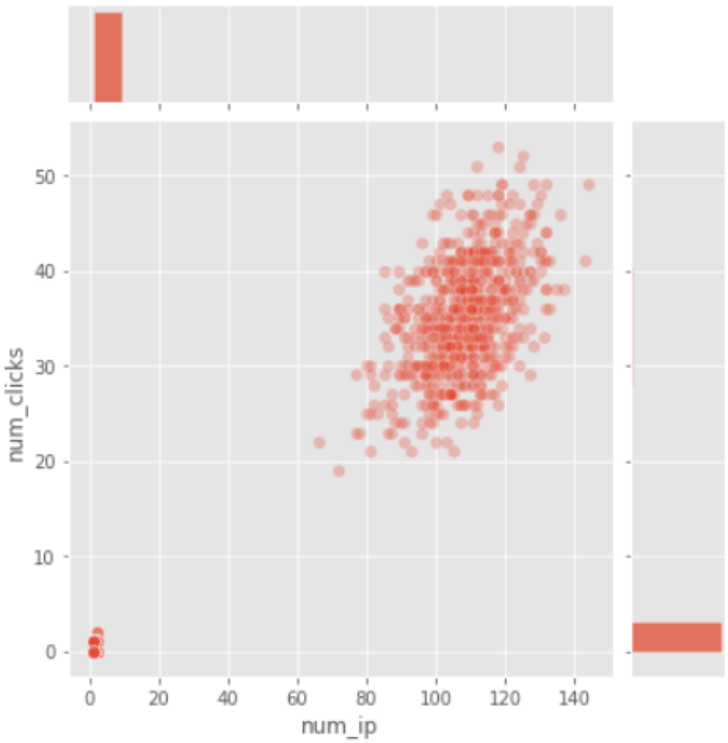

Figure 4: Each dot is an user within a time window represented by his number of clicks and the number of associated IP addresses.

1.5 Removing suspicious patterns on the test set

We filter the activities based on the three mentioned rules on the test set. Table sums up the evolution of the global CTR. Fig 5 shows the evolution of CTR remains between 5% and 15%.

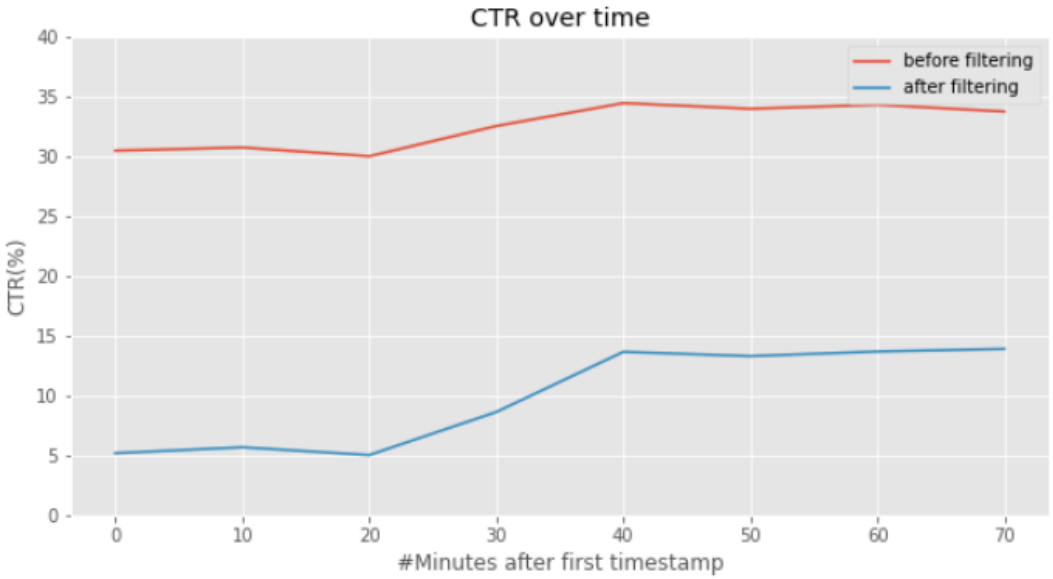

Figure 5: Comparison of the evolution of the CTR before and after filtering suspicious patterns.

2 Implementation

After the analysises, we implemented the fraud detection logic with Flink in Java.

2.1 General pipeline

The general pipeline is shown in Fig 6. Events are outputed from Kafka. They are deserialized into activities (instance of Activity class). Since our fraudulent patterns are about IP addresses and UIDs, activites are keyed by these entities before being processed (computing average reaction time of users, counting events...). The pipeline is divided into two parts. The first part join these data into a single stream and sink it to an ElasticSearch-Kibana stack in order to feed the prepared dashboard. The second part aims to filter and output suspicious events to a file.

We keep these two part separated since the project's task was about writing suspicious events to a file. We worked on the creation of a dashboard in order to give the possibility to monitor events easily. Fig 8 shows a screenshot of the dashboard.

CAVEAT: Sometimes, the mean reaction time cannot be calculated because no click has been recorded within a given time window and user. That is the reason why we fixed an arbitrary value of 10000 seconds as a mean time reaction (which is above the maximum value) so that they won't be filtered in our system and can be easily retrieved.

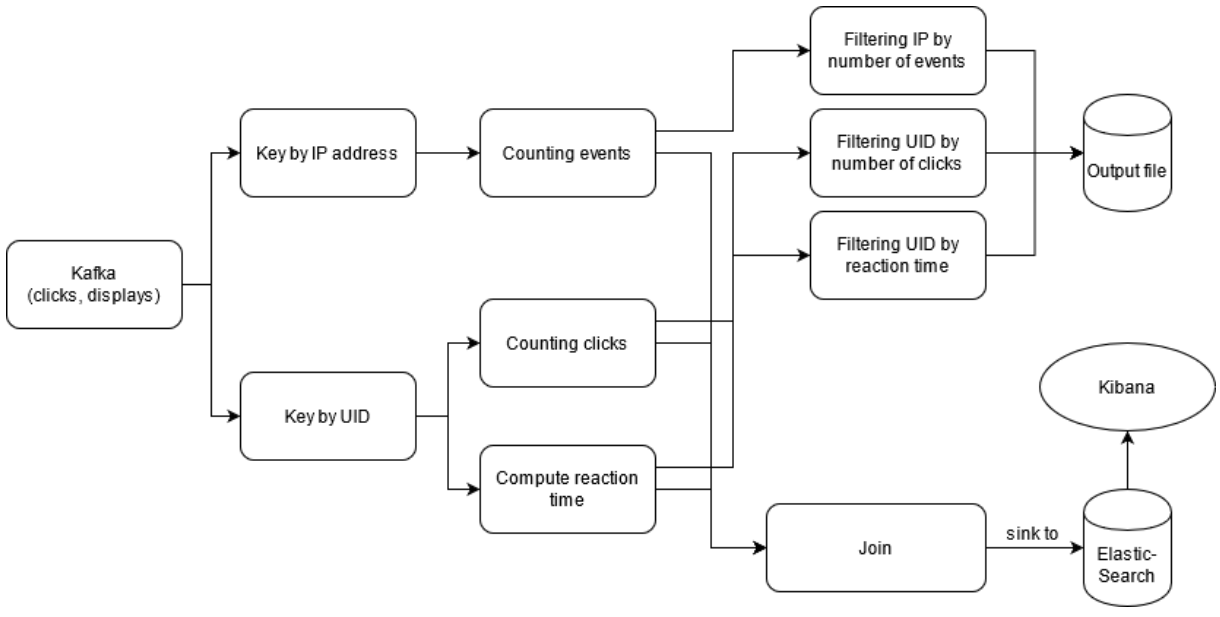

Figure 6: General data pipeline and processing schema from the capture of raw data (clicks and displays) to the output of clean data and suspicious activities.

2.2 Window

We choose a window of 10 minutes to accumulate enough events to analyze. We tolerate a lateness of 1 second and choose a tumbling window. Since the window is large enough, there should be enough data to make the system work correctly even without some late events that are caught after 1 second of lateness.

2.3 Structure

First, we defined a class Activity to represent the events. With this done, we created and modified the deserialisation schema of the kafka source to enable better control over the events and their form in the Flink application. We also set a Alert class which is a subclass of Activity that represent fraudulent activity and we created a Alert Sink to output the fraudulent events in the Standard output and in a dedicated file as well. We created a ActivityStat class (subclass of Activity) as well to output statistics about the activities to Elastic Search and Kibana.

The class diagram for the complete application is the following one. There are classes for the functions computing the fraud detection logic that we will talk about in the below section.

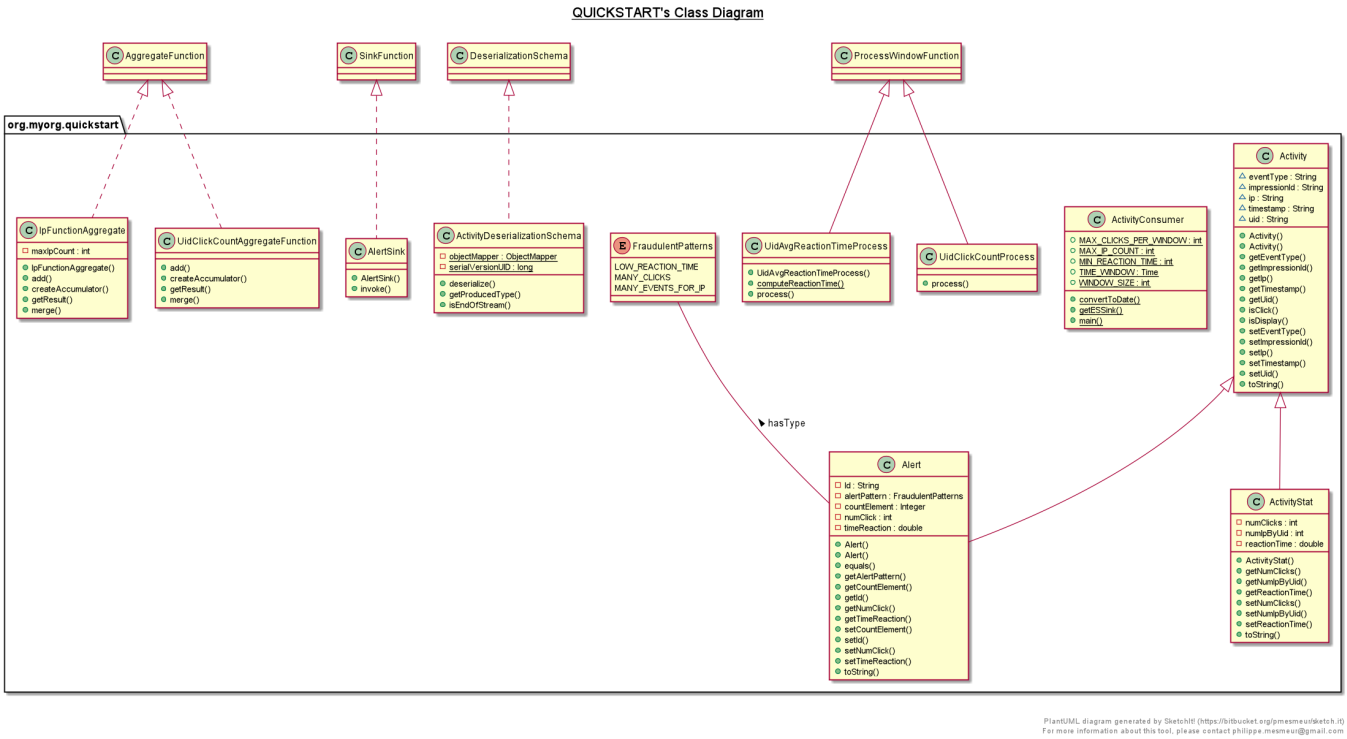

Figure 7: Class diagram of the application.

2.4 Functions

As derived from the analysises made, we implemented the logic by using the following window operators : AggregateFunction and ProcessWindowFunction. For the number of events per IP and the number of clicks per Uid, we could use a stateless operator as the logic did not impose to have access to the whole window informations. We choose to use an Aggregate functions and thus defined the UidClickCountAggregateFunction and an IPFunctionAggregate. Theses classes all derived from AggregateFunction<I,O,A>. I being tuples whith the appropriate number of fields from the events. For the Uid average reaction time process, we needed to have a stateful operator as we compute the average reaction time of an Uid on the whole window. We thus choose to use a ProcessWindowFunction which gets a tuple in input with the appropriate fields from the events.

In the ActivityConsumer class which contains the main() method, we start by enabling event time by setting a WatermarkStrategy directly on the kafka source, the timestamps of the Kafka records themselves are used for the timestamp. Then, we add the kafka source with a ActivityDeserializationSchema to retrieve the fields properly. There are DataStream<Alert> to create Alert objects which are then put in a AlertSink and there is an DataStream<ActivityStat> which is obtained by joining activity stream with ipCount, this datastream has an Elastic Search Sink.

2.5 Tests

To make sure everything worked as intended and that there were no regression or other types of software engineering problems, we implemented tests for all the functions derived from the analyses. The tests over the aggregate functions were pretty straightforward as they are stateless operators. To test the average click time for a particular UID, we had to do some other stuff since the ProcessWindowFunction is kind of a stateful operator.
First, we wanted to use test harnesses (OneInputStreamOperatorTestHarness) which can be used to push records and watermarks into our stateful functions but the demarch was unsucessful as there were import problems with maven that we didn't manage to resolve. So, we ended up recreating a small data source by hand and testing the stateful operator directly with a sink.

Finally, on top of the unit test set, we made a useful visualisation tool to make sure the CTR was normalizing with the fraudulent pattern implemented.

3 Dashboard

The dashboard is used to monitor the stream of activities and the CTR over time. It gives the possibility to evaluate the variation of CTR by modifying the parameters of the rule-based method (increasing number of clicks, ...) which can be seen in Fig 8 with the control panel.

Figure 8: Dashboard made to monitor activities in real time.

Data are sink at the end of the time window which means the dashboard needs to be refreshed (Refresh button on the top right corner). Visualisation which computes an aggregation such as the average number of clicks per UID is computed over the time interval set from Kibana and can be edited by clicking on the calender icon on the top-right corner.

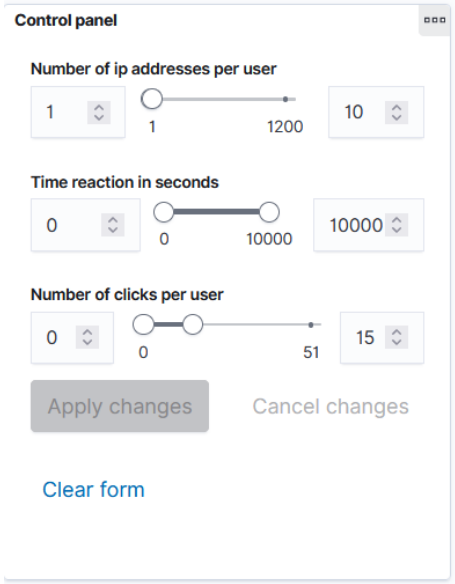

Figure 9: The control panel allows to modify the rule-based method's parameters to impact the CTR over time. Actually, it impacts all visualisations.

Figure 10: Displays the CTR over a time window of one minute.

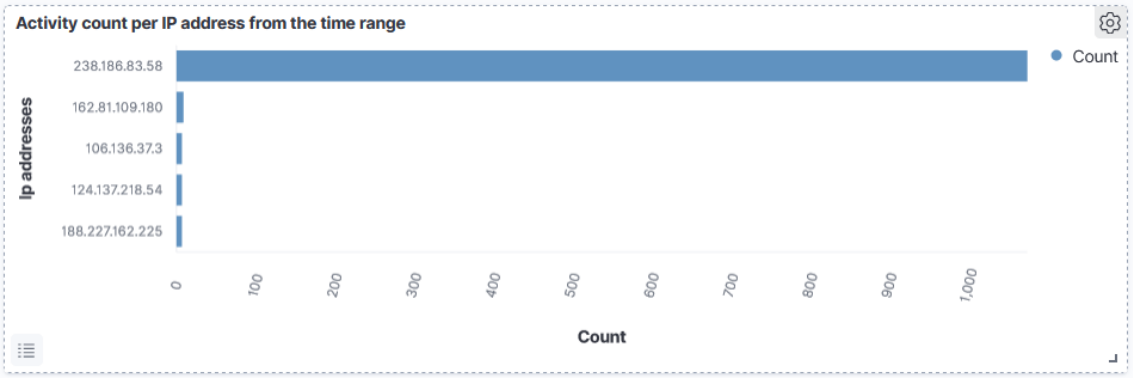

Figure 11: Shows the top IP addresses over Kibana's time interval that maximize the count.

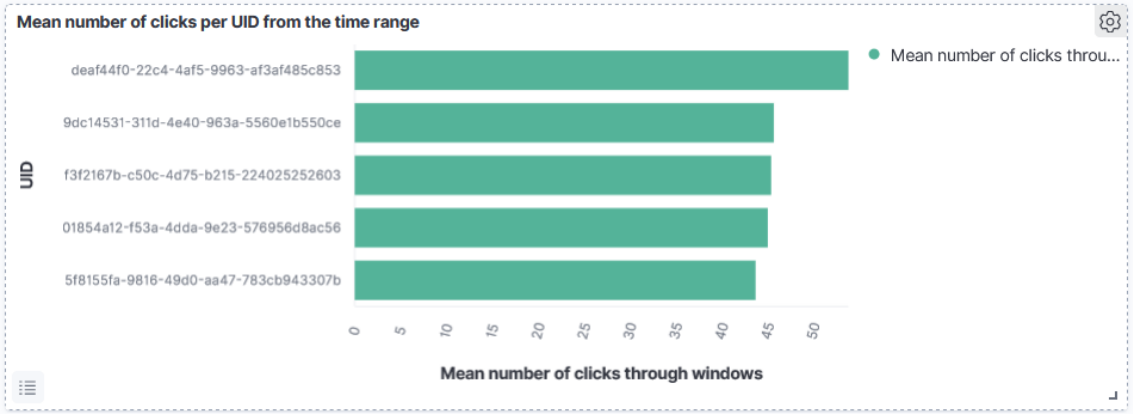

Figure 12: Shows the top UID over Kibana's time interval with the highest average click value : Flink application calculates the number of clicks within a time window, since Kibana's time interval can be larger than Flink's time window, we choose the display the average number of clicks per UID over Flink's time windows within that lies on Kibana's time interval.

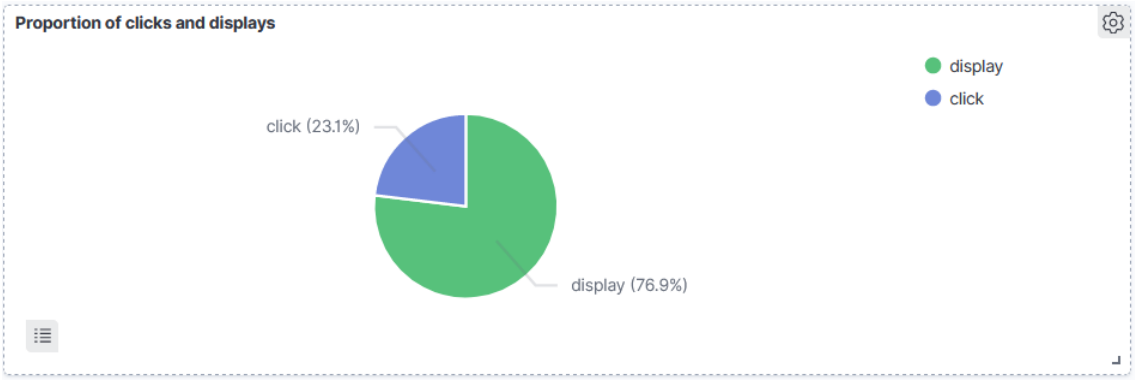

Figure 13: Displays the proportion of clicks and displays accumulated in Kibana's time interval.

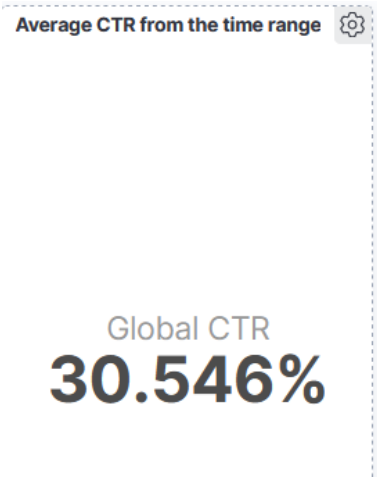

Figure 14: Calculates the average value of the means CTR over windows of one minutes that falls in Kibana's time interval.

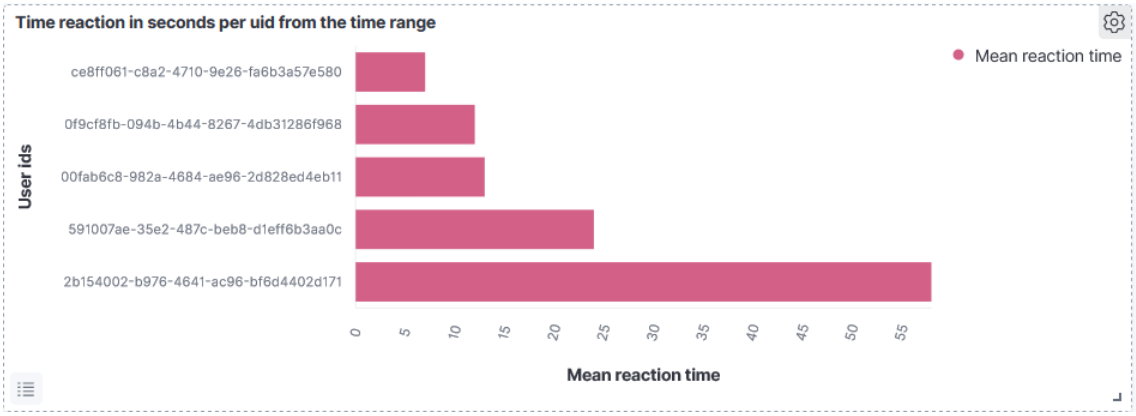

Figure 15: Shows top users with the lowest average reaction time. It is calculated by taking the mean value of the average time reaction of users received from Flink application.