KTH Royal Institute of Technology
Stockholm

School of Electrical Engineering and
Computer Science

Data-Intensive Computing - ID2221

Analyzing recent changes on Wikipedia using Kafka and Spark Streaming

Author
Emil Ståhl

Author
Selemawit Fsha

October 24, 2021

Project - ID2221

Emil Ståhl and Selemawit Fsha Nguse

October 3, 2021

1 Introduction

In this project, we analyze recent changes made on Wikipedia.org including articles written in all available languages. Data is obtained from Wikimedia's stream available at `https://stream.wikimedia.org/v2/stream/recentchange`. With our thorough analysis, we are able to get insight about metrics such as **articles changed per minute**, **articles changed by user**, **number of changes to an article in a given time span**, and the **percentage of changes made by bots**.

2 Dataset

The stream available at `https://stream.wikimedia.org/v2/stream/recentchange` contiously updates with the most recent changed articles along with metadata such as **title**, **user**, **bot**, and **timestamp**. The data is structured in JSON format. The complete schema is:


```
{
    "$schema": "/mediawiki/recentchange/1.0.0",
    "meta": {
        "uri": "https://commons.wikimedia.org/wiki/File:ISS020-E-39833_-_View_of_Earth.jpg",
        "request_id": "5382ba2a-4d6c-4e2b-9334-38e78f047060",
        "id": "b56809a1-a550-48ad-9237-892a6d4bf309",
        "dt": "2021-10-17T21:37:36Z",
        "domain": "commons.wikimedia.org",
        "stream": "mediawiki.recentchange",
        "topic": "eqiad.mediawiki.recentchange",
        "partition": 0,
        "offset": 3371393091
    },
    "id": 1791566123,
    "type": "log",
    "namespace": 6,
    "title": "File:ISS020-E-39833 - View of Earth.jpg",
    "comment": "",
    "timestamp": 1634506656,
    "user": "Askeuhd",
    "bot": false,
    "log_id": 317367914,
    "log_type": "upload",
    "log_action": "upload",
    "log_params": {
        "img_sha1": "nm1p8x5se8dst4sr16yhmeow11bsoi7",
        "img_timestamp": "20211017213736"
    },
    "log_action_comment": "uploaded &quot;[[File:ISS020-E-39833 - View of Earth.jpg]]&quot;",
    "server_url": "https://commons.wikimedia.org",
    "server_name": "commons.wikimedia.org",
    "server_script_path": "/w",
    "wiki": "commonswiki",
    "parsedcomment": ""
```

3 Tools

To achieve the desired results, we make use of the following tools:

- Kafka (message broker for reading and writing recent changes)
- Akka (for dealing with concurrent GET request)
- Spark Structured Streaming (to process real-time data)
- Maven (build automation tool)
- Docker (to package the application and get a standardized executable component)
- HBase (Distributed database)

4 Methodology

Here we present the methodology used and the functionality of our code:

4.1 Producer.scala

In this file, we utilize Akka for making concurrent GET request to the stream available at Wikimedia's URL. We establish a connection to our Kafka server running at port 9092. Next we create a KafkaProducer object with our created properties. The producer is flushed to get rid of previous messages. Akka streams provides a RestartSource, RestartSink and RestartFlow for implementing the so-called exponential backoff supervision strategy, starting an operator again when it fails or completes, each time with a growing time delay between restarts. This pattern is useful when the operator fails or completes because some external resource is not available and we need to give it some time to start-up again. Next, we use Source.fromFutureSource which streams the elements of the given future source once it successfully completes. If the future fails the stream is failed. Then we do a foreach loop on the source object creating a producer record with our topic and data to send. The data is finally sent.

4.2 Consumer.scala

In Consumer.scala, we establish a connection to our Kafka server and subscribing to the topic we created. Next, we create a schema with the desired fields we want to make use of in our analysis. Next, we need to convert our Kafka topic input stream value, that has a binary type, into a meaningful dataframe. In this case we selected the value, as we are not interested in the other fields provided by the kafka stream. There is a nested structure JSONtostructs that contains all the JSON fields. We need to select the embedded values:

```
val aggregation = initial.select(from_JSON($"value", schema).alias("tmp")).select("tmp.*")
```

We now want to define the window size and watermark to be able to do data analysis on a specific time span and deal with late messages:

```
.withWatermark("timestamp", "1 minutes")
.groupBy(window($"timestamp", "1 minute", "1 minute"), $"user", $"bot")
```

Next we do our analysis:

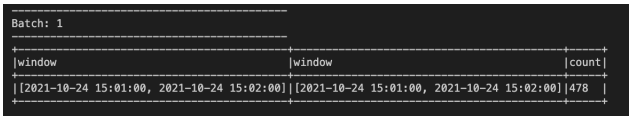
```
val botsCount = valueJson
  .withWatermark("timestamp", "1 minutes")
  .groupBy(window($"timestamp", "1 minute", "1 minute"))
  .agg(
    sum(when($"bot" === true, lit(1))).as("is_bots"),
    sum(when($"bot" === false, lit(1))).as("non_bots")
  )
```

5 Results

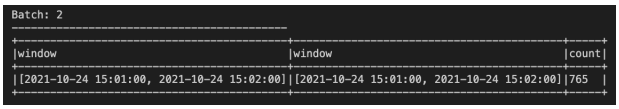
In this section, we present our results obtained from the data analysis.

5.1 Number of changed articles within 1 minute

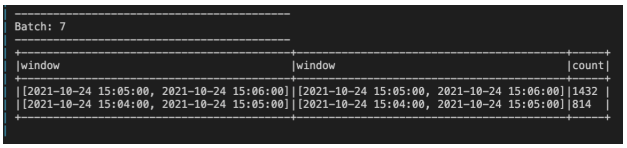
When first starting the consumer it will report the number of changed articles from when it started to analyse the stream until the window reaches 1 minute. Since the consumer is most likely started not started exactly at the start of a new window this first window will not display correct information.



The printout of batches is structured as follows: The program prints out new information every 4 seconds regardless of the windowing. However, the information about the correct window will be updated. As shown below, we can see that in the first window the number of articles changed increased by 287 within 4 seconds between batch 1 and batch 2.

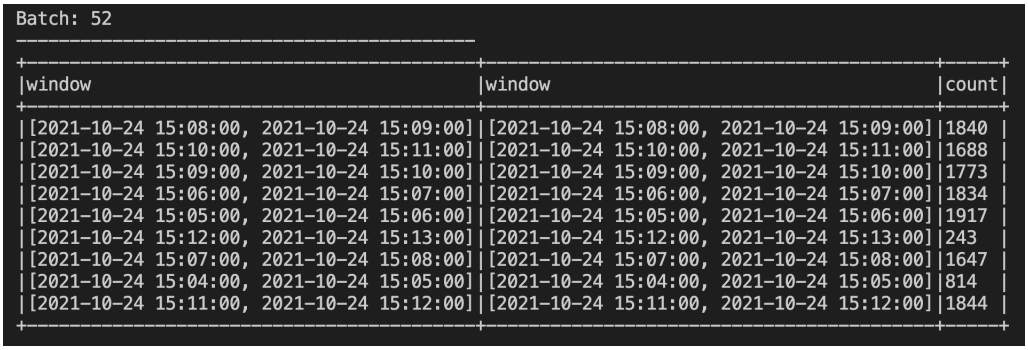


In batch 7, we see that the first window has ended and a new one started. As mentioned, the first window did not count articles changed in a whole minute, therefore the count for the first window is incorrect.



```
-------------------------------------------
Batch: 7
-------------------------------------------
+------------------------------------------+------------------------------------------+-----+
|window                                    |window                                    |count|
+------------------------------------------+------------------------------------------+-----+
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|1432 |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|814  |
+------------------------------------------+------------------------------------------+-----+
```

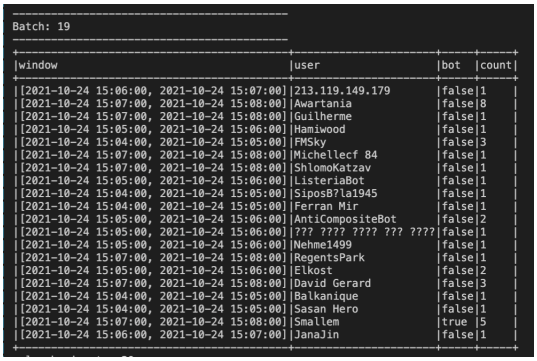
In batch 52, we can see that a number of windows has passed and that the number of changed articles per minute is steadily around 1,500 - 2,000. As previously mentioned, the first batch should be neglected. Note also that the current window of 15:12 - 15:13 is currently being processed and updated.



```
Batch: 52
-------------------------------------------
+------------------------------------------+------------------------------------------+-----+
|window                                    |window                                    |count|
+------------------------------------------+------------------------------------------+-----+
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|1840 |
|[2021-10-24 15:10:00, 2021-10-24 15:11:00]|[2021-10-24 15:10:00, 2021-10-24 15:11:00]|1688 |
|[2021-10-24 15:09:00, 2021-10-24 15:10:00]|[2021-10-24 15:09:00, 2021-10-24 15:10:00]|1773 |
|[2021-10-24 15:06:00, 2021-10-24 15:07:00]|[2021-10-24 15:06:00, 2021-10-24 15:07:00]|1834 |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|1917 |
|[2021-10-24 15:12:00, 2021-10-24 15:13:00]|[2021-10-24 15:12:00, 2021-10-24 15:13:00]|243  |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|1647 |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|814  |
|[2021-10-24 15:11:00, 2021-10-24 15:12:00]|[2021-10-24 15:11:00, 2021-10-24 15:12:00]|1844 |
+------------------------------------------+------------------------------------------+-----+
```

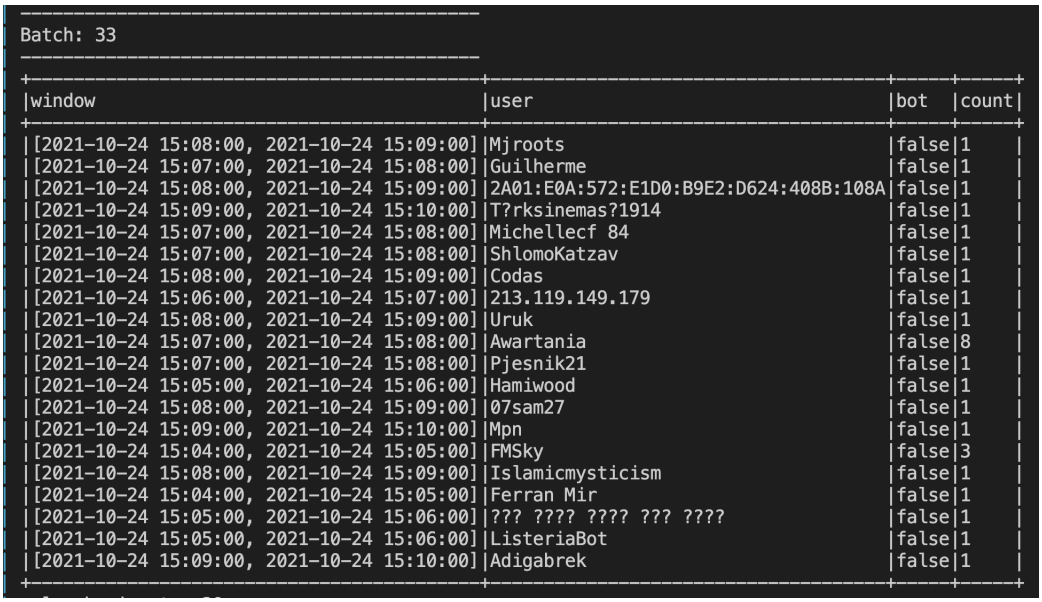
5.2 Changes per user

Here, we analyze the number of changes made on a per user basis. In batch 19 we see the first 20 rows of changes per user. The data is not sorted on the number of changes.



```
-------------------------------------------
Batch: 19
-------------------------------------------
+------------------------------------------+--------------------+-----+-----+
|window                                    |user                |bot  |count|
+------------------------------------------+--------------------+-----+-----+
|[2021-10-24 15:06:00, 2021-10-24 15:07:00]|213.119.149.179     |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Awartania           |false|8    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Guilherme           |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|Hamiwood            |false|1    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|FMSky               |false|3    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Michellecf 84       |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|ShlomoKatzav        |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|ListeriaBot         |false|1    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|SiposB7la1945       |false|1    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|Ferran Mir          |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|AntiCompositeBot    |false|2    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|??? ???? ???? ??? ??|false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|Nehme1499           |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|RegentsPark         |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|Elkost              |false|2    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|David Gerard        |false|3    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|Balkanique          |false|1    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|Sasan Hero          |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Smallem             |true |5    |
|[2021-10-24 15:06:00, 2021-10-24 15:07:00]|JanaJin             |false|1    |
+------------------------------------------+--------------------+-----+-----+
```

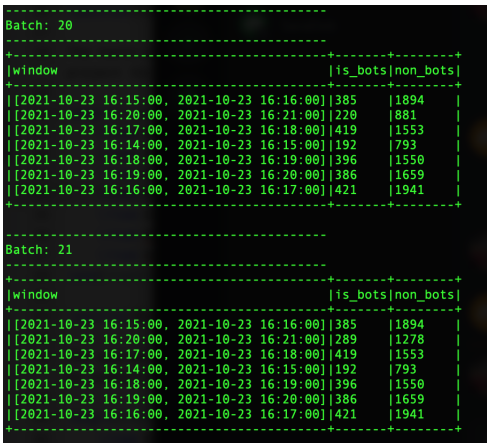
In batch 33, we see another set of users. The table can be expanded to show all users.



```
-------------------------------------------
Batch: 33
-------------------------------------------
+------------------------------------------+--------------------------------------+-----+-----+
|window                                    |user                                  |bot  |count|
+------------------------------------------+--------------------------------------+-----+-----+
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|Mjroots                               |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Guilherme                             |false|1    |
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|2A01:E0A:572:E1D0:B9E2:D624:408B:108A|false|1    |
|[2021-10-24 15:09:00, 2021-10-24 15:10:00]|T?rksinemas?1914                      |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Michellecf 84                         |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|ShlomoKatzav                          |false|1    |
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|Codas                                 |false|1    |
|[2021-10-24 15:06:00, 2021-10-24 15:07:00]|213.119.149.179                       |false|1    |
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|Uruk                                  |false|1    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Awartania                             |false|8    |
|[2021-10-24 15:07:00, 2021-10-24 15:08:00]|Pjesnik21                             |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|Hamiwood                              |false|1    |
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|07sam27                               |false|1    |
|[2021-10-24 15:09:00, 2021-10-24 15:10:00]|Mpn                                   |false|1    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|FMSky                                 |false|3    |
|[2021-10-24 15:08:00, 2021-10-24 15:09:00]|Islamicmysticism                      |false|1    |
|[2021-10-24 15:04:00, 2021-10-24 15:05:00]|Ferran Mir                            |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|??? ???? ???? ??? ????                |false|1    |
|[2021-10-24 15:05:00, 2021-10-24 15:06:00]|ListeriaBot                           |false|1    |
|[2021-10-24 15:09:00, 2021-10-24 15:10:00]|Adigabrek                             |false|1    |
+------------------------------------------+--------------------------------------+-----+-----+
```

5.3 Bots vs. non-bots

Here, we calculate the number of changes made by bots and non-bots. However, it should be noted that even if an user is classified as non-bot it may not be correct since multiple usernames contain "bot" in their name despite not being classified as one. It can be seen that the number is approximately the number as changed articles which is correct



```
-------------------------------------------
Batch: 20
-------------------------------------------
+------------------------------------------+-------+--------+
|window                                    |is_bots|non_bots|
+------------------------------------------+-------+--------+
|[2021-10-23 16:15:00, 2021-10-23 16:16:00]|385    |1894    |
|[2021-10-23 16:20:00, 2021-10-23 16:21:00]|220    |881     |
|[2021-10-23 16:17:00, 2021-10-23 16:18:00]|419    |1553    |
|[2021-10-23 16:14:00, 2021-10-23 16:15:00]|192    |793     |
|[2021-10-23 16:18:00, 2021-10-23 16:19:00]|396    |1550    |
|[2021-10-23 16:19:00, 2021-10-23 16:20:00]|386    |1659    |
|[2021-10-23 16:16:00, 2021-10-23 16:17:00]|421    |1941    |
+------------------------------------------+-------+--------+


-------------------------------------------
Batch: 21
-------------------------------------------
+------------------------------------------+-------+--------+
|window                                    |is_bots|non_bots|
+------------------------------------------+-------+--------+
|[2021-10-23 16:15:00, 2021-10-23 16:16:00]|385    |1894    |
|[2021-10-23 16:20:00, 2021-10-23 16:21:00]|289    |1278    |
|[2021-10-23 16:17:00, 2021-10-23 16:18:00]|419    |1553    |
|[2021-10-23 16:14:00, 2021-10-23 16:15:00]|192    |793     |
|[2021-10-23 16:18:00, 2021-10-23 16:19:00]|396    |1550    |
|[2021-10-23 16:19:00, 2021-10-23 16:20:00]|386    |1659    |
|[2021-10-23 16:16:00, 2021-10-23 16:17:00]|421    |1941    |
+------------------------------------------+-------+--------+
```

5.4 Bot percentage

With the data from the following DataFrame, we can calculate the percentage of bots making changes on Wikipedia.

```
-------------------------------------------
Batch: 4
-------------------------------------------
+------------------------------------------+------+--------+-------------------+
|window                                    |is_bots|non_bots|bots_percentage    |
+------------------------------------------+------+--------+-------------------+
|[2021-10-24 21:14:00, 2021-10-24 21:15:00]|455    |1216    |0.2722920406941951 |
|[2021-10-24 21:13:00, 2021-10-24 21:14:00]|284    |1124    |0.20170454545454544|
|[2021-10-24 21:15:00, 2021-10-24 21:16:00]|null   |1       |null               |
+------------------------------------------+------+--------+-------------------+
```

5.5 Writing to Kafka

Since multiple aggregations are not supported, we can write the aggregated stream to Kafka and then do a second level of aggregation on the stream from Kafka.

```
def write(df: DataFrame, topic: String): Unit = {
    val stream = df
    .writeStream
    .outputMode("update")
    .foreachBatch { (batchDF: DataFrame, batchId: Long) =>
      batchDF.collect().foreach { row =>
        val record = new ProducerRecord[String, String](topic, row.toString())
        //println("Writing stream " + topic + " to kafka")
        this.producer.send(record)
      }
    }
    .start()
  }
```

5.6 Write to HBase

The results from the data analysis can be stored on a distributed Database like HBase:

```
def write(df: DataFrame, tableName: String): Unit = {
      val stream = df
      .writeStream
      .outputMode("update")
      .foreachBatch { (batchDF: DataFrame, batchId: Long) =>
        batchDF.collect().foreach { row =>
          val put = new Put(row.getAs[String]("key").getBytes())
          put.addColumn("cf".getBytes(), "title".getBytes(), row.getAs[String]("title").getB
          put.addColumn("cf".getBytes(), "user".getBytes(), row.getAs[String]("user").getByt
          put.addColumn("cf".getBytes(), "bot".getBytes(), row.getAs[Boolean]("bot").toStrin
```

```
        put.addColumn("cf".getBytes(), "timestamp".getBytes(), row.getAs[String]("timestam
        put.addColumn("cf".getBytes(), "id".getBytes(), row.getAs[String]("id").getBytes()
        table.put(put)
      }
    }
    .start()
}
```

6 Discussion

The results of this work may be of interest for organizations such as Wikipedia to know what changes are being made on their site. For example, if some specific article id is edited multiple times with in a given time, it could be irregular. That's someone could be just playing around with that article for miscellaneous purposes. Other purposes can be to just get an insight into how much data is manipulated in a given time.

7 Encountered problems

During the project, we encountered a number of problems. First of all, the chosen dataset was not optimal for doing interesting data analysis due to lack of data. This made it difficult to perform extensive analysis. We also had problems writing to HBase where our dependencies did not work as intended. Also, due to that a Dataframe cannot have multiple aggregations, for example first counting number of changes for each and then calculating the maximum changes, we could not do all analysis that we wanted.

8 How to run

To run the project, make sure the following dependencies are installed:

- Docker including docker compose
- Maven
- Make

In the project directory, open a terminal window and type **make create-network** followed by **make run-producer** and let it start up. When it prints "starting to produce messages" move on to the consumer. Open another terminal window and type **make run-consumer**, let it build for 1 minute, when it displays the schema it has successfully initialized. Now it will print out the results of the data analysis.