Apache Beam

Harsimran Kaur
Department of Applied Modelling, course: Big Data, Trent University

INTRODUCTION

Apache Beam is an open-source, unified programming model designed to process large-scale, unbounded, and batch data processing pipelines. It provides a simple yet powerful API for building and executing data processing pipelines that can run on a variety of execution engines such as Apache Flink, Apache Spark, and Google Cloud Dataflow.

It is useful for write a logic only once and run it on different platforms like spark, flink, data flow etc., which gives the flexibility to port the same logic to multiple execution platform.

Provides a unified programming model for batch and stream processing. Supports multiple languages such as Java, Python, Go, and others. Supports multiple execution engines, allowing for portability of pipelines across different systems.

Provides advanced features such as windowing, triggers, and side inputs. Offers a flexible and scalable architecture for handling large volumes of data. It is used to reduce latency of individual results.

AIM

The aim of the project is to implement the beam pipeline to read data and apply windowing on it.

INTRODUCING APACHE BEAM

The Unified Apache Beam Model

The easiest way to do batch and streaming data processing. Write once, run anywhere data processing for mission-critical production workloads.

Beam is used in ETL process i.e. Extract, Transform and Load data to further save it as a large, central repository called a datawarehouse.

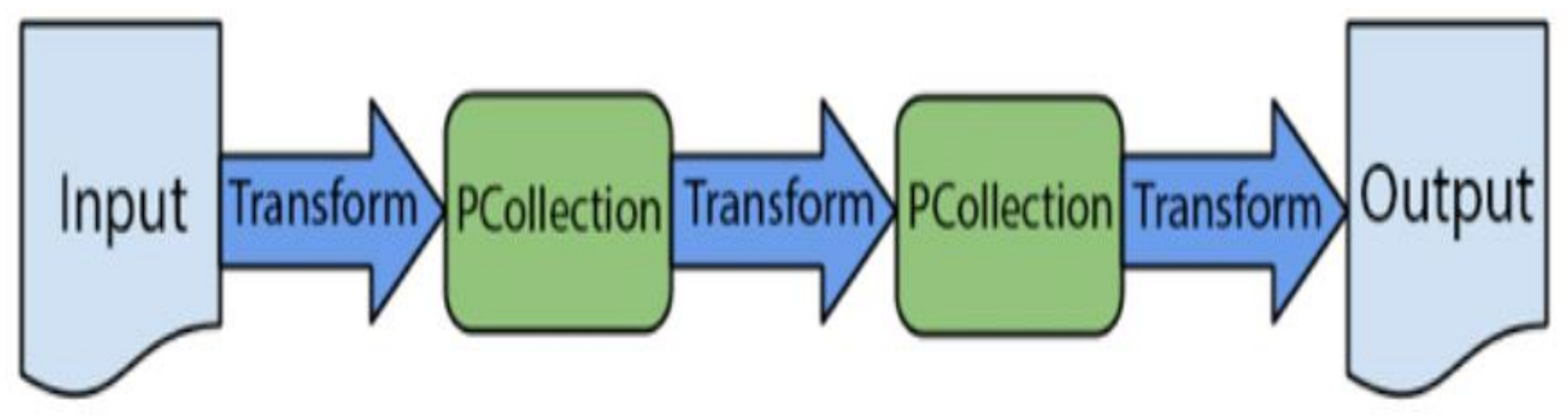
METHOD

A typical Beam driver program works as follows:

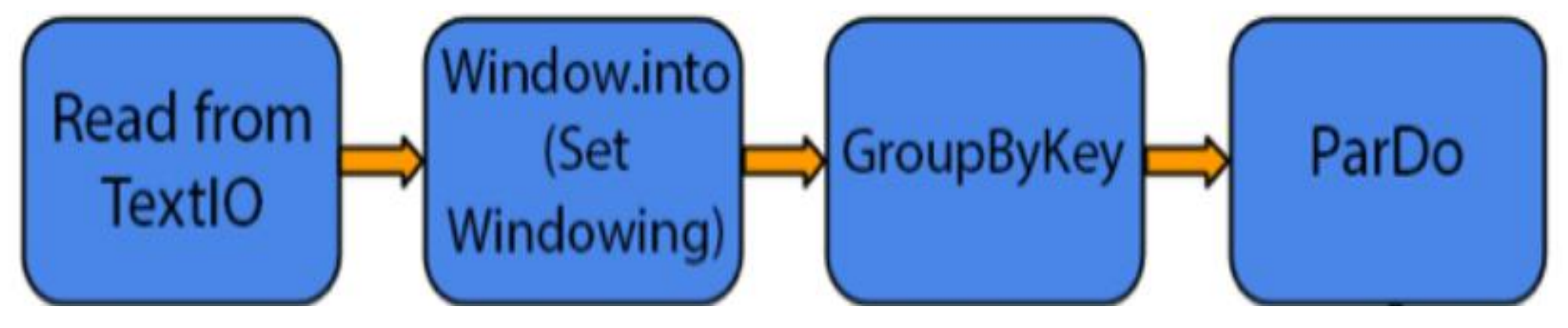
- Create a pipeline object
- Create an initial Pcollection
- Apply PTransforms
- Use IOs to write the transformed Pcollection to external source
- Run the pipeline using the Runners.

Basic Terminologies in Beam:

- Pipeline : A pipeline is a user-constructed graph of transformations that defines the desired data processing operations.
- PCollection : A PCollection is a data set data stream. The data that a pipeline processes is part of a PCollection.
- PTransform : A PTransform (or transform) represents a data processing operation in your pipeline.
- Aggregation : Aggregation is computing a value from multiple input, example: summation.
- User-defined function (UDF) : Some Beam operations anow you to run user-defined code as a way to configure the transform.
- Runner : A runner runs a Beam pipeline using the capabilities of the chosen data processing engine. Examples include: Apache Fink, Cloud Dataflow, etc.



Functional Model for Apache Beam



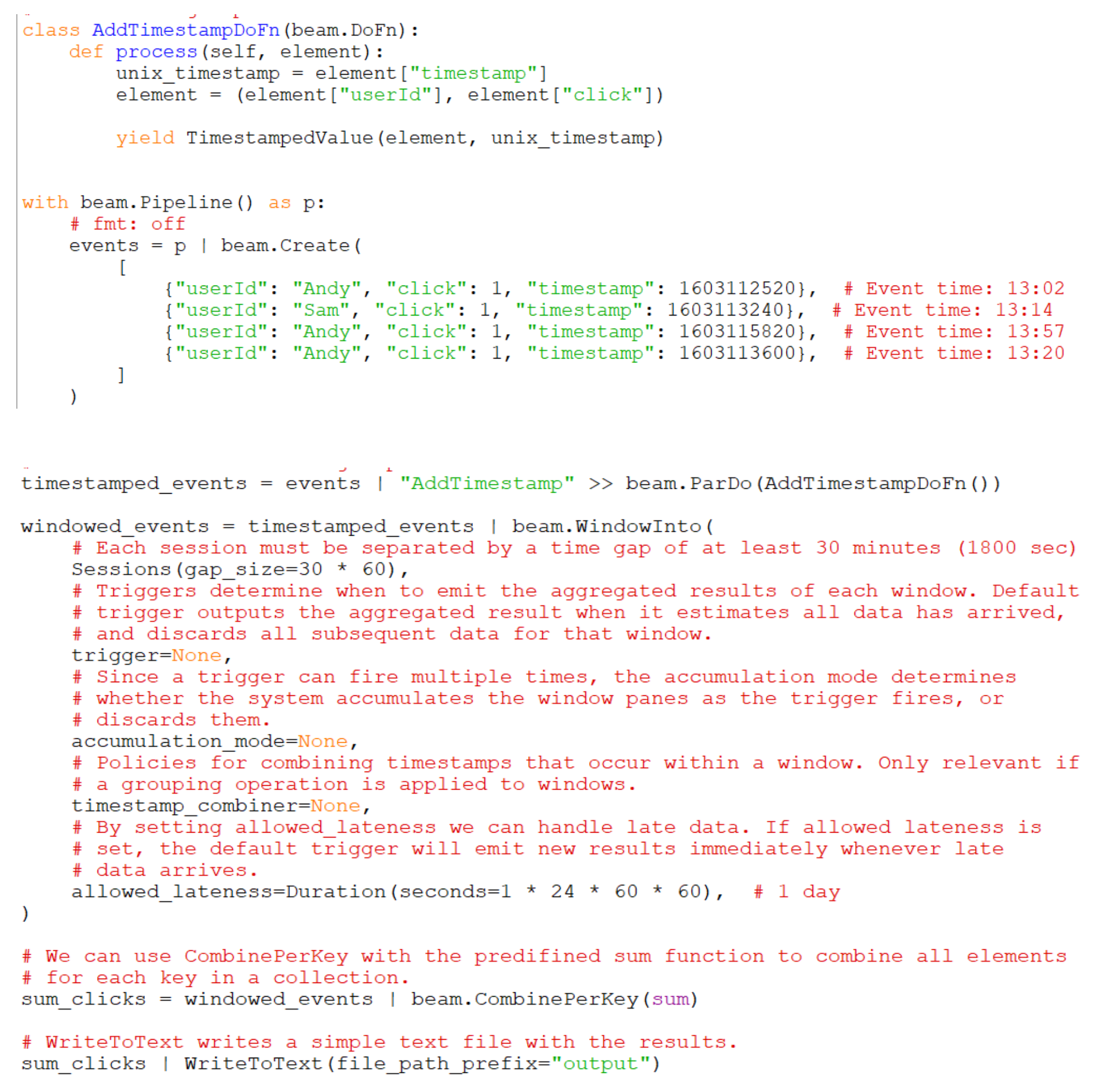
Data Flow Model for Apache Beam

RESULT

The Apache Beam pipeline is applied for collection of data which here is the clicks made by each user. Which is then transformed into tuple to isolate users and assigned a unique tuple id.

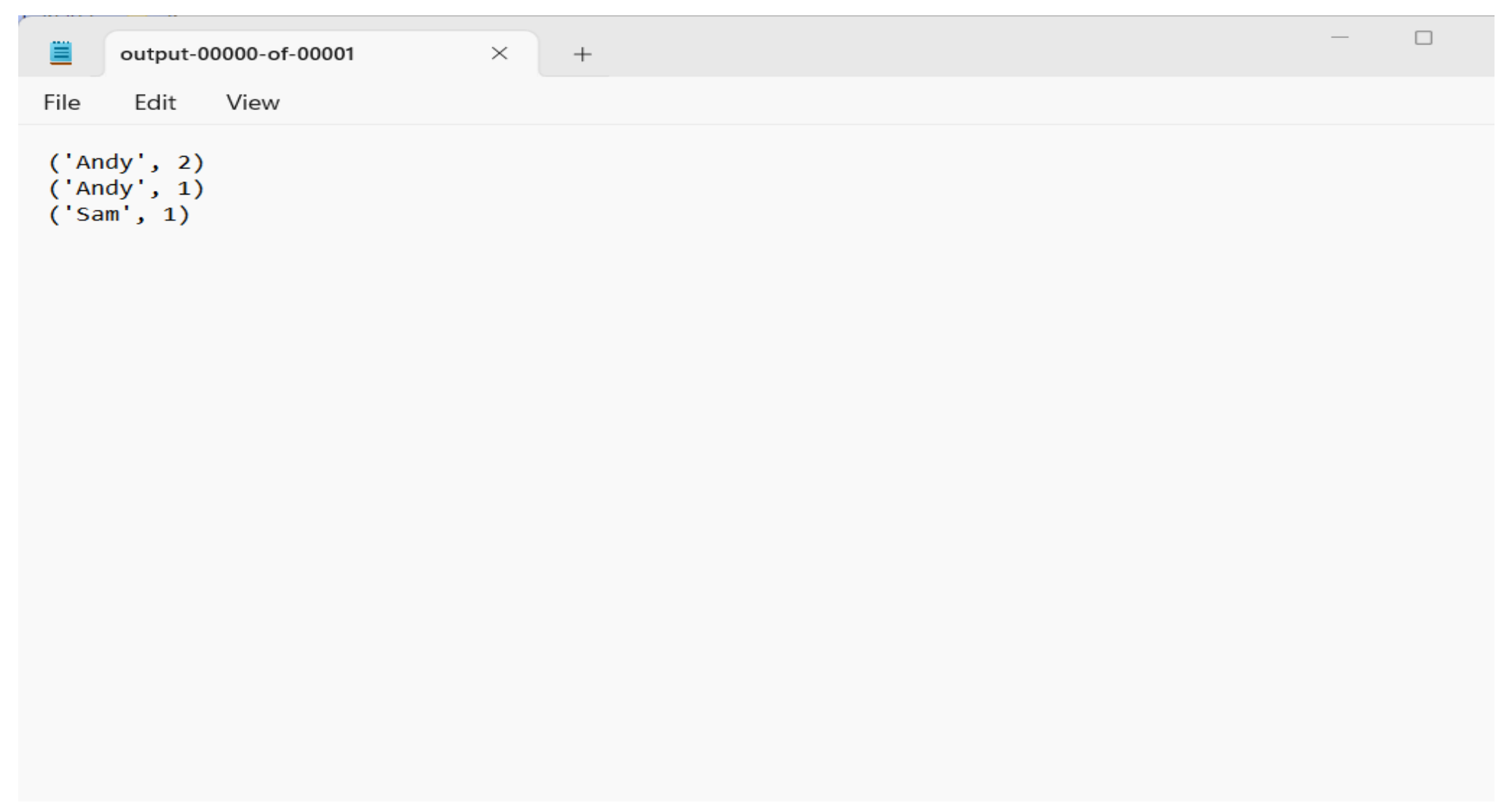
They are then stamped in to a window of size 30 minutes. And number of clicks are then added based on the window for each user.

Each window is space if the user is not responding for more than 30 minutes.



```
class AddTimestampDoFn(beam.DoFn):
    def process(self, element):
        unix_timestamp = element["timestamp"]
        element = (element["userId"], element["click"])

        yield TimestampedValue(element, unix_timestamp)


with beam.Pipeline() as p:
    # fmt: off
    events = p | beam.Create(
        [
            {"userId": "Andy", "click": 1, "timestamp": 1603112520},  # Event time: 13:02
            {"userId": "Sam", "click": 1, "timestamp": 1603113240},  # Event time: 13:14
            {"userId": "Andy", "click": 1, "timestamp": 1603115820},  # Event time: 13:57
            {"userId": "Andy", "click": 1, "timestamp": 1603113600},  # Event time: 13:20
        ]
    )


timestamped_events = events | "AddTimestamp" >> beam.ParDo(AddTimestampDoFn())

windowed_events = timestamped_events | beam.WindowInto(
    # Each session must be separated by a time gap of at least 30 minutes (1800 sec)
    Sessions(gap_size=30 * 60),
    # Triggers determine when to emit the aggregated results of each window. Default
    # trigger outputs the aggregated result when it estimates all data has arrived,
    # and discards all subsequent data for that window.
    trigger=None,
    # Since a trigger can fire multiple times, the accumulation mode determines
    # whether the system accumulates the window panes as the trigger fires, or
    # discards them.
    accumulation_mode=None,
    # Policies for combining timestamps that occur within a window. Only relevant if
    # a grouping operation is applied to windows.
    timestamp_combiner=None,
    # By setting allowed_lateness we can handle late data. If allowed lateness is
    # set, the default trigger will emit new results immediately whenever late
    # data arrives.
    allowed_lateness=Duration(seconds=1 * 24 * 60 * 60),  # 1 day
)

# We can use CombinePerKey with the predifined sum function to combine all elements
# for each key in a collection.
sum_clicks = windowed_events | beam.CombinePerKey(sum)

# WriteToText writes a simple text file with the results.
sum_clicks | WriteToText(file_path_prefix="output")
```

The screenshot of the code depicting the use of beam pipeline to collect the data
The beam.create creates the data that will be transformed
The timestamp event is a subset of beam.ParDo (Parallel processing) to collect the click time stamp of all users simultaneously
The beam.windowinto slicing the data into windows of 30 minutes gaps from previous activity (here is it clicks)
Beam.CombinePerKey(sum) sums up the activity of each user thus showing us how many clicks were done by each user in the windows
WritetoText function write the summed output into a file named: "output-00000-of-00001", output is shown below

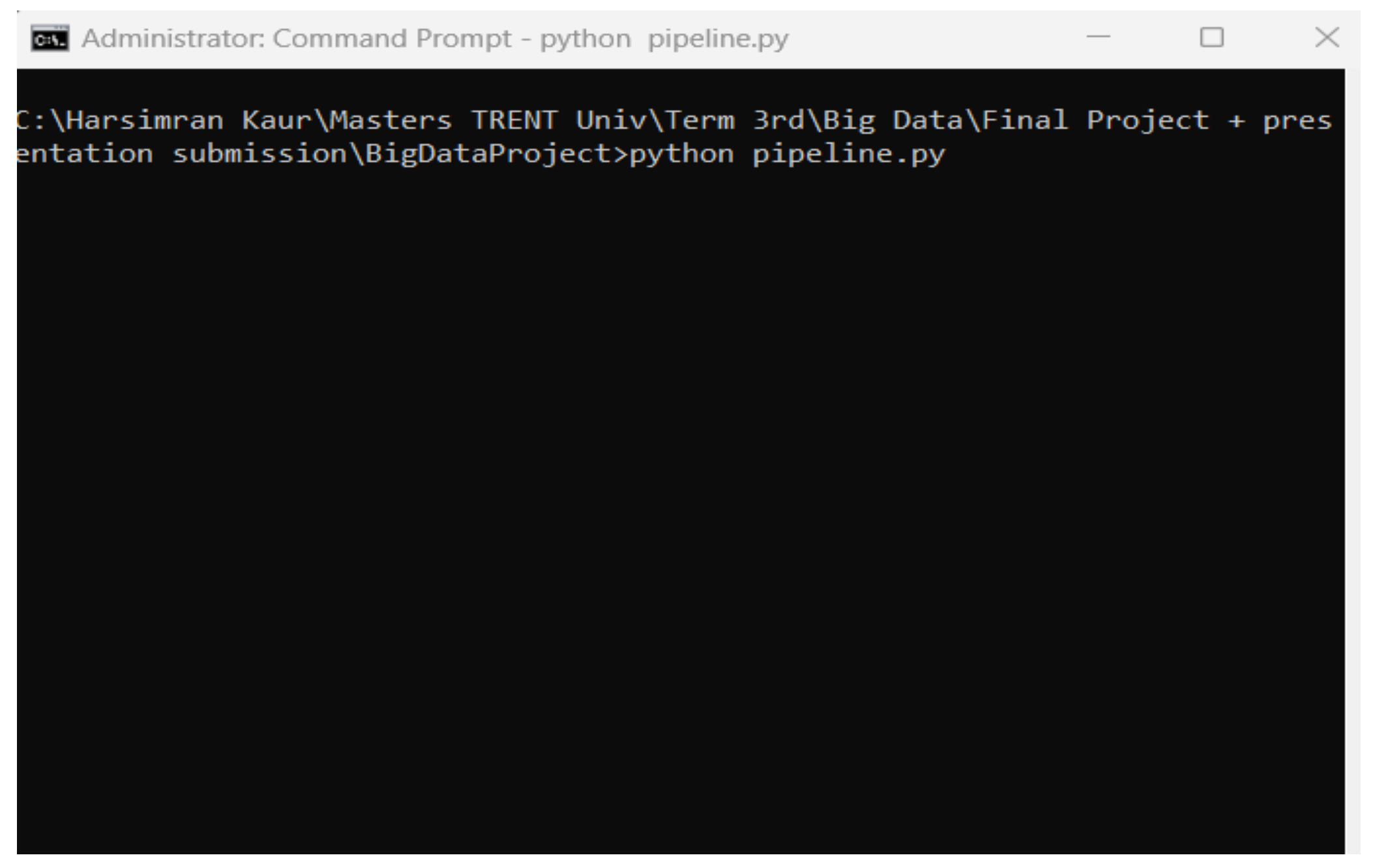


output-00000-of-00001

```
('Andy', 2)
('Andy', 1)
('Sam', 1)
```

Output of the project shows completion of the pipeline and the screenshot above shows the transformed windowed and summed data.

PRINTING



Administrator: Command Prompt - python pipeline.py

```
C:\Harsimran Kaur\Masters TRENT Univ\Term 3rd\Big Data\Final Project + pres
entation submission\BigDataProject>python pipeline.py
```

The above screenshot of the command prompt shows how I run my pipeline code. Once we the run the command the output file as shared is adjoining result section is produced.

CONCLUSIONS

Beam pipeline can be used to fetch data of Grocery list at supermarkets, for calculating traffic onto a website etc., without having to stop sessions of users. The data is as when generated and there is no time space in which the data is deemed as completed. This also updates data as it comes hence reducing data latency and users can see the changes as they use the services.

ACKNOWLEDGEMENTS

https://beam.apache.org/documentation/
https://www.youtube.com/watch?v=yZUe4th9gwY&t=608s

Link to My Video presentation:
https://youtu.be/ddU583bJbU8

beam