SDS Assignment 4

Siddharth Menon

November 2025

1 Data Schema and Processing Logic

1.1 Event Schema

Both Spark and Flink processors consume events from the Kafka topic `ad-events-2`. The schema used for parsing these events is as follows:

- `user_id` (string): Unique ID of the user.
- `page_id` (string): The page on which the ad was shown.
- `ad_id` (string): Identifier for the advertisement.
- `ad_type` (string): Category or type of the advertisement.
- `event_type` (string): Either `view` or `click`.
- `event_time` (long): Event timestamp in milliseconds (generated by the producer).
- `ip_address` (string): User IP address.
- `timestamp` (long): Kafka timestamp recorded at ingestion time.

This schema is used to construct a structured DataFrame in Spark and a source table in Flink. The payload timestamp, `event_time`, is converted into a proper SQL `TIMESTAMP` column called `event_ts` to support event-time processing.

1.2 Kafka Sink Output Format

Before writing results back to Kafka, Spark formats each row as a JSON structure containing:

- `campaign_id`
- `window_start`, `window_end`
- `views`, `clicks`

- `ctr`
- `max_kafka_ts`

These are serialized using:

```
to_json(named_struct(...)) AS value
```

The sink topic is `results`, consumed later for throughput and latency measurements.

1.3 Parallelism

This was tested with parallelism=1 and parallelism=4 which is the number of partitions in the input Kafka topic.

1.4 Summary

This schema-rich, event-time-aware pipeline allows both Spark and Flink processors to:

1. Ingest raw ad events.
2. Parse and convert timestamps.
3. Apply watermarks to manage out-of-order data.
4. Perform windowed CTR aggregation.
5. Output cleaned, aggregated results to Kafka.

These processed results were later used to benchmark throughput and latency for Spark vs Flink, shown in the observation plots.

2 System Architecture - Spark

The streaming data pipeline consists of three main components: a data generator, a Kafka cluster, and a Spark Structured Streaming processor.

The data generator continuously produces ad events, which are pushed to the Kafka topic `ad-events-2`. The Spark processor reads the events from this topic, parses the incoming JSON data, and assigns event-time timestamps with watermarks to handle out-of-order events. It then performs 10-second tumbling window aggregations to compute the number of views and clicks for each `campaign_id`.

Finally, the aggregated results, including CTR and the maximum Kafka timestamp per window, are written back to the Kafka topic `results` for downstream consumption.

The flow is as follows:

A dataframe is first created from the raw JSON data which applies a watermark of 1 second. It then splits it into 2 dataframes for aggregating view and click events in 10 second windows (view_counts and click_counts).

A final dataframe is then created using view_counts and click_counts to compile their data and calculate CTR for the incoming data.

3 System Architecture - Flink

The Flink streaming data pipeline also consists of three main components: a data generator, a Kafka cluster, and a Flink SQL processor.

The data generator continuously produces ad events, which are pushed to the Kafka topic `ad-events-2`. The Flink SQL processor reads the events from this topic by defining a source table in Flink with the appropriate schema and a 1-second watermark on the event timestamp (`event_ts`) to handle out-of-order events.

A 10-second tumbling window aggregation is then performed using Flink SQL, computing the number of clicks and views per `campaign_id`. The Click-Through Rate (CTR) is calculated per window as:

Finally, the aggregated results, including CTR and the maximum production timestamp per window, are written to the Kafka topic `results`.

The flow is as follows:

A source table is created in Flink that reads the raw JSON events from Kafka and converts the event time into a timestamp column. Flink SQL then computes windowed aggregates directly over this table, calculating clicks, views, and CTR for each `campaign_id` in 10-second tumbling windows. The results are inserted into the sink Kafka table `results` for downstream consumption.

4 Observations

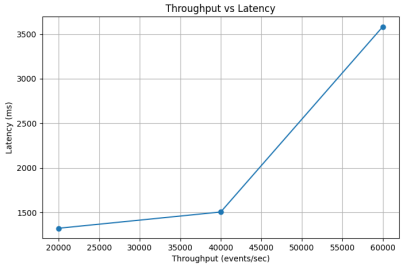

Figure 1: Latency: Flink

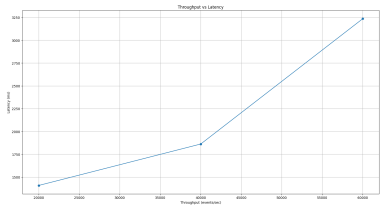

Figure 2: Latency: Spark

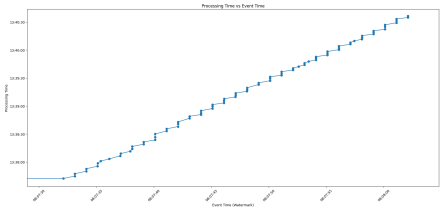

Figure 3: Watermark progression - Spark - normal event generation

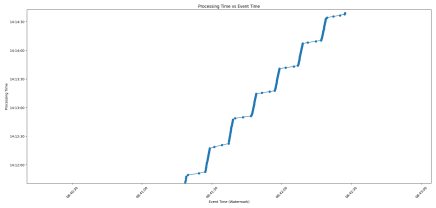

Figure 4: Watermark progression - Spark - MMPP