13th ACM International Conference on Distributed and Event-based Systems (DEBS), June 24-29, 2019, Darmstadt, Germany

Generating Reproducible Out-of-Order Data Streams

Philipp M. Grulich
grulich@tu-berlin.de

Jonas Traub
jonas.traub@tu-berlin.de

Asterios Katsifodimos
a.katsifodimos@tudelft.nl

Tilmann Rabl
tilmann.rabl@hpi.de

Sebastian Breß
sebastian.bress@dfki.de

Volker Markl
volker.markl@tu-berlin.de

Experiments with Out-of-Order Streams

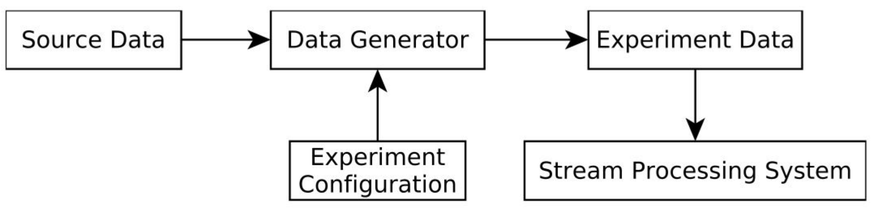
We provide a <u>scalable</u> data stream generator, which introduces <u>configurable</u> out-of-orderness in real-world data streams. This enables <u>reproducible</u> and <u>realistic</u> experiments.

- Handling out-of-order data streams is a key feature of modern stream processing systems [1].
- Research on the support of out-of-order stream processing requires **<u>reproducible</u>**, **<u>scalable</u>**, and **<u>configurable</u>** experiments on out-of-order data streams. For example, research on efficient window aggregation [2,3,4].

The evaluation of out-of-order capabilities is hard:

- Public real-world datasets do not reflect all aspects of out-of-order streams (e.g., different delays and fractions of out-of-order tuples).
- Experiments without real-world data can lead to unrealistic results.

Architecture Overview



A general out-of-order experiment data generator:

- Introduce out-of-order data to real-world input data sets.
- Generic configuration of out-of-orderness to enable full parameter exploration.
- Reproducible generation of experimental data with configurable out-of-orderness.

Generator Configuration

```
1  "dataSource": {
2    "file": $path$,
3    "seperator": ","|";"|"\t",
4    "time": {
5      "timeIndex": $field$,
6      "sourceTimeUnit": "ps"|"ns"|"ms"|"s"
7    }
8  },
9  "experimentDataConfigurations": [
10   {
11     "targetOutOfOrderFactor": [0-100],
12     "minDelay": 0,
13     "maxDelay": 2000,
14     "delaySeed": $seed$
15   }
16 ]
```

- **Configurable aspects of out-of-order streams:**
  - Fraction of out-of-order tuples (How many tuples are out-of-order?).
  - Minimal/maximal event delay (How late are out-of-order tuples?).
  - Out-of-order delay distribution (How are delays distributed?).

References

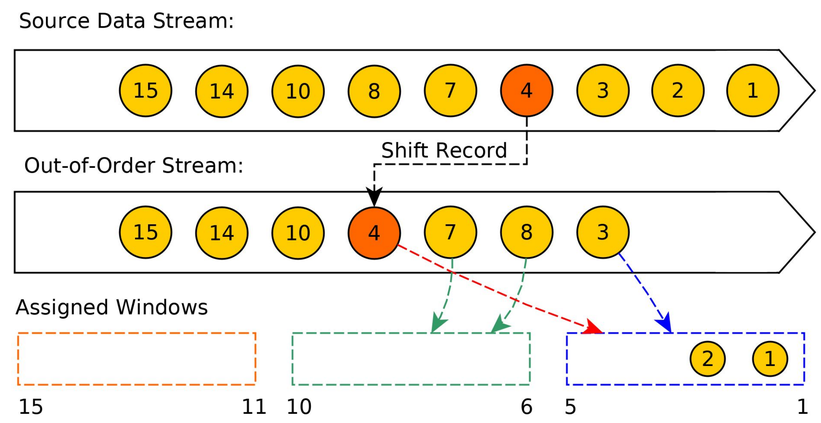
[1] Tyler Akidau, Robert Bradshaw, et al. VLDB 2015.
The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing.

[2] Jonas Traub, Philipp M. Grulich, Alejandro R. Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. EDBT 2019.
Efficient Window Aggregation with General Stream Slicing.

[3] Jonas Traub, Philipp M. Grulich, Alejandro R. Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. ICDE 2018.
Scotty: Efficient Window Aggregation for Out-of-Order Stream Processing.

[4] Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. arXiv preprint 2018.
Sub-O (log n) Out-of-Order Sliding-Window Aggregation.

Adding out-of-orderness to Data Streams



Goal:

- The generation of out-of-order tuples has to be deterministic and must not change query results.

Naive Solution:

- Generating out-of-order data by adding a random delay to the event time of certain tuples → changes temporal data distribution.

Our Approach:

- Shift ingestion times of source tuples and keep original event times.

Generator Algorithm

Step 1. Preprocessing:

- Analyze the out-of-orderness of the source data stream to take this knowledge into account for data generation.

$maxTs \leftarrow 0$;
for *record in sourceFile* **do**
  if $record.ts < maxTs$ **then**
    `RegisterDelayedRecord(`*record*`)`;
  end
  $maxTs \leftarrow record.ts$;
end

Step 2. Generation of out-of-order ingestion time:

- If an in-order tuple becomes an out-of-order tuple, we add a random delay to its ingestion time (based on the configured distribution).
- Finally, we sort the data set by ingestion time.

for *record in recordBuffer* **do**
  if $record.ts < maxTs$ **then**
    $record.ingestionTime \leftarrow record.ts$;
  else
    $delay \leftarrow$ `createDelay()`;
    $record.ingestionTime \leftarrow record.ts + delay$;
  end
end
`sort(`*recordBuffer*, $r \rightarrow r.ingestionTime$`)`;

Step 3. Ingestion to the stream processing system:

- Only ingest records if ingestion time is reached.

for *record in recordBuffer* **do**
  if $record.ingestionTime > now$ **then**
    `wait(`*record.ingestionTime-now*`)`;
  end
  `emit(`*record*`)`
end

Technische Universität Berlin

German Research Center for Artificial Intelligence

TU Delft

Open Source Repository
TU-Berlin-DIMA/out-of-order-datagenerator