# Spark Streaming Assignment

D'Ambrosi Denis

January 11, 2024

1. **Explain the difference between the continuous operator model and Sparks Discretized Streams. How can recovery be implemented in each model?**
   The continuous operator model treats data as a continuous flow and as such it applies the operators continuously as new data is retrieved. Each operator keeps an internal state that gets updated upon the retrieval of a new datapoint. The main recovery mechanisms are recovery by replication (based on making nodes redundant) and upstream backup (keeping a backup of recent states to quickly instantiate a new node in case of failure). In any case, stragglers are not handled, but this is an inherent problem of online execution. Discretized Streams instead handle data in mini-batches (RDDs) determined by time intervals. Recovery mechanisms are the same of standard RDDs (in particular checkpointing and RDD Lineage). Treating streams as mini-batches allows some flexibility for out-of-order arrival and stragglers.

2. **How can aggregation benefit from defining an invertible function?**
   When aggregation is performed on overlapping windows, an invertible function allows to efficiently compute the new aggregate by removing (via the inverse) the elements of the last aggregate that are no longer in scope and introducing only the new ones. Without an inverse we would need to recompute from scratch the aggregate for each window considered.

3. **Explain the consistency semantics of streaming systems. What's the difference between at most once, at least once, and exactly once? How can exactly once be realized?**
   In a stream processing system there may be a chain of computation with a failing node. The approach for handling such failure to resume the pipeline between the failed operator and the next one may depend on the context of the application, but will fall into one of three categories:

   - *At-most-once* semantics require that failed computations simply discard the failure and continue the pipeline. This can clearly lead to data loss, but not data replication.

- *At-least-once* semantics require that messages should be re-delivered in case of failure. This ensures to avoid data loss, but can lead to data replication if ack messages are lost.
- *Exactly-once* semantics require that each message is delivered exactly once in any case. This can be implemented through distributed snapshots for rollbacks in case of failures or transaction logs for deduplication. It ensures to avoid both data loss and data replication but requires more complex systems and computational resources.

4. **What is the difference between the output modes of unbounded tables (complete, append, update). What are the consequences for late data in case of event time windows?**
   The difference between the various output modes the which kind of rows are written to the output sink at the end of each interval:
   - *Complete mode* requires that the whole table gets rewritten. Late data within the watermark will update the complete table during the next interval.
   - *Append mode* allows to write only the rows of the last interval (clearly assumes that previous data does not change). The window does not get revised in the case of late data, so it must arrive before the output of the table to get included.
   - *Update mode* enables to show which rows are changed from the previous interval as diff (allows data to mutate). Late data will cause updates to the result table.

5. **What is a watermark? How is it decided whether an event is below the watermark?**
   A watermark is a criterion that is used to avoid waiting indefinitely for late data: by defining a maximum threshold time $T_\Delta$, we can discard any event $e$ that arrives at time $T_e$ belonging to interval $i$ if the observed maximum event time before the beginning of $i$, $T_i$, satisfies the following relation:

$$T_i > T_e + T_\Delta$$