Fault Tolerance

What is Fault Tolerance?

- Property that enables a Spark system to continue operating properly in the event of the failure of some of its components
- Mechanisms in spark include Checkpoints and HDFS
- Understanding the Semantics is important for becoming proficient with streaming.

Fault Tolerance of Spark RDDs

RDDs are the basic unit of Spark Streaming. Their fault tolerance can be summarized with the three following points

1. An RDD is an immutable, deterministically re-computable, distributed dataset.
2. Any partition of an RDD lost to a worker node failure can be re-computed from the original fault-tolerant dataset using the lineage of operations.
3. Assuming all transformations are deterministic, the final transformed RDD’s data will always be the same.

Spark with *Streaming* is a bit riskier….

DStream fault tolerance is not always guaranteed.

This means that in order to achieve fault tolerance, multiple spark executors must replicate the data among at least 2 worker nodes in the spark cluster.

Two kinds of data (based on reaction to failure):

1. Data that survives failure of a single worker node because it is replicated after being received, so a copy of it exists on one of the other nodes.
2. Data that is *buffered for replication, but not actually replicated* at the time of the failure. When this happens, the only way to recover this data is to get it again from the source.

Three Types of Guarantees

Categories of guarantees of protection against driver failure (described by how many times a record is processed by the system):

1. ***At most once*:** which means Each record will be either processed once or not processed at all.
2. ***At least once*:** which means Each record will be processed one or more times. This is stronger than *at-most once* as it ensure that no data will be lost. But there may be duplicates.
3. ***Exactly once*:** Each record will be processed exactly once - no data will be lost and no data will be processed multiple times. This is obviously the strongest guarantee of the three.

Three Steps to Processing Streams

1. *Receiving the data*: The data is received from sources using Receivers or otherwise.
   - Different input sources provide different guarantees ranging from *at-least once* to *exactly once*.
2. *Transforming the data*: The received data is transformed using DStream and RDD transformations.
   - All data that has been received will be processed *exactly once*
3. *Pushing out the data*: The final transformed data is pushed out to external systems like file systems, databases, dashboards, etc.
   - Output operations by default ensure *at-least once* semantics, but users can implement their own transaction mechanisms to achieve *exactly-once* semantics

Fault Tolerance for Receivers

Two main types of input receivers:

1. ***Reliable Receivers,*** which acknowledge reliable sources only after ensuring that the received data has been replicated. If such a receiver fails, the source will not receive acknowledgment for the buffered data. Therefore, if the receiver is restarted, the source will resend the data, and no data will be lost due to the failure.
2. ***Unreliable Receivers***, which do not send acknowledgment and therefore can lose data when they fail due to worker or driver failures.

Additional Strategies for Avoiding Loss

- *Write ahead logs* save the received data to fault-tolerant storage. With the write ahead logs enabled and reliable receivers, there is zero data loss. In terms of semantics, it provides an *at-least once* guarantee.
- Output operations, like `foreachRDD()`, also have *at-least once* semantics. That is, the transformed data may get written to an external entity more than once in the event of a worker failure.