Discretized Streams

Record-at-a-time processing models raise several challenges such as

1. Fault tolerance
2. Consistency
3. Unification with batch processing

They key idea behind DStreams is to treat streaming computation as a series of deterministic batch computations across small time intervals. The two advantages of this are

1. Consistency is well defined
2. Can integrate with batch processing systems

To meet optimal latency requirements, the intermediate states of DStreams are stored in memory. But this is expensive due to the cost of data replication into a memory-based key/value store. Instead, RDDs are used as a storage abstraction that can rebuild lost data through lineage without replication.

Failure recovery is done by parallelising recovery, where each node in the cluster works to recompute a part of the lost node's RDDs, resulting in faster recovery.

A DStream groups together a series of RDDs and lets the user manipulate them through operators. Stateless operators such as map and stateful operators such as aggregations over windows can be used.

DStream Operators

DStreams provide two kinds of operators

1. Transformations
    a. Produce new DStreams from one or more parent streams
    b. Can be Stateless or Stateful
2. Outputs
    a. Write data to external systems

DStreams support the same stateless transformations available in typical batch frameworks such as map, reduce, groupBy and join.

Stateful operators are used to work on multiple intervals

1. Windowing
    a. window() operator
    b. Inefficient as it recomputes for every window
2. Incremental aggregation
    a. reduceByWindow()
    b. More efficient that windowing
3. Time skewed joins
    a. Join streams against its own RDDs

Output operators such as save, foreach and write provide output writes to external systems for DStreams.