Apache Spark Streaming

Introduction - Motivation

Apache Spark Streaming is a layer working on top of Spark.

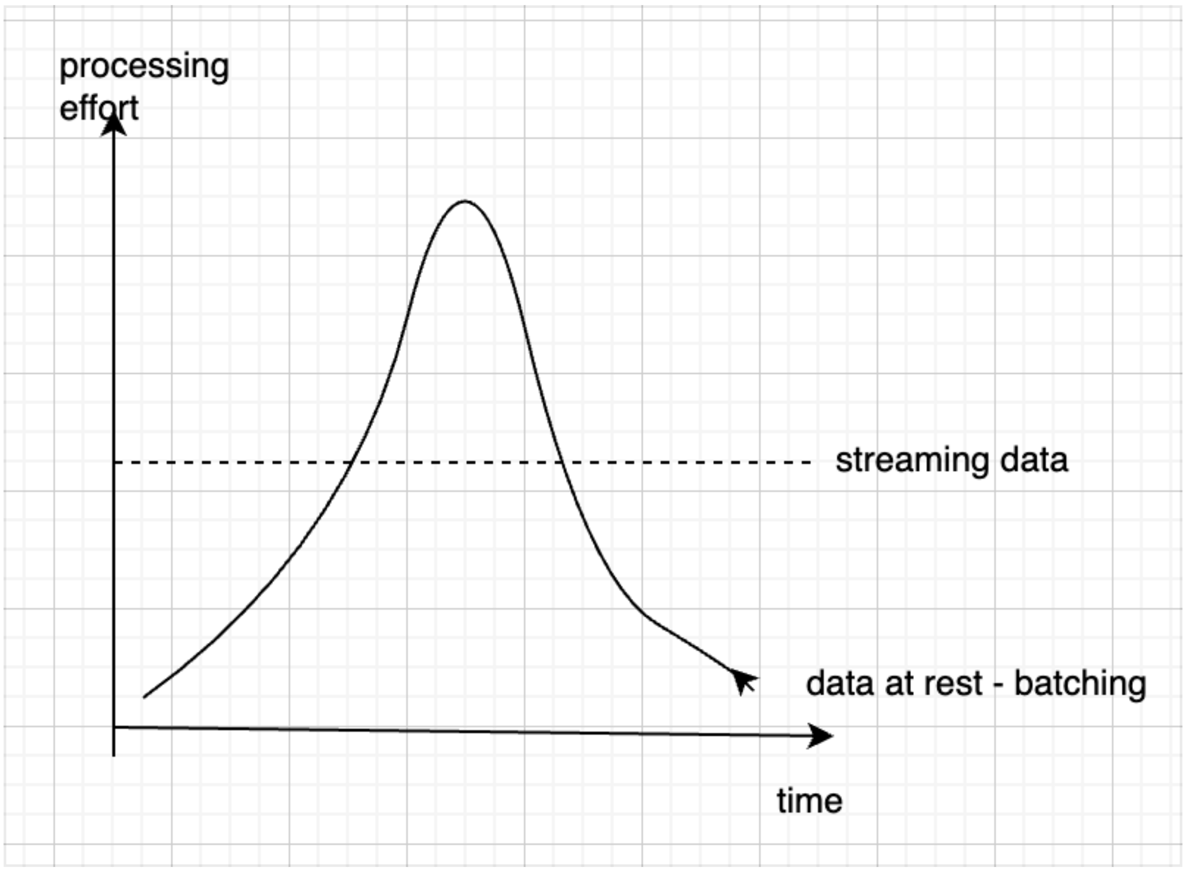
It is highly relevant to today's technological needs of businesses, due to the fact that data are required with ever-decreasing latency, and streaming data sources can accommodate that.

However, it is also the case that streaming data tend to have enormous size, they are unbounded, and they require special handling.

Apache Spark Streaming offers an efficient way to handle this. Via its application, we can move the processing stage of the data to the moment they arrive to our systems. This spreads workloads out more evenly over time, yielding more consistent and predictable consumption of resources.

Definitions:

- A Data Stream := is an unbounded sequence of data. It is produced by a source ("server" through its 'sockets') and consumed by a destination ("client").
- Steam processing := is low latency processing and analysis of streaming data. It divides continuously flowing input of data into discrete chunks for processing.
- Apache Streaming := a type of data processing engine that is designed with infinite (unbounded) data sets in mind. In other words, it is an ongoing mode of data processing applied to the unbounded type of data. It differs from 'batch' processing which used to be the norm in parallel processing up to fairly recently. Even though intra-day ETL workflows achieved lowering latency levels by a lot, every time a batch runs (after potentially being at rest across a cluster of machines) it leads to very 'spiky' demands in CPU/memory/etc. In contrast, by streaming we spread the processing of the same amount of data in smaller intervals and handle it in a more stable and flexible way.
- A socket := a communication link between two applications - we need the IP of the server from which data is read, and the port of the DStream to emit data into



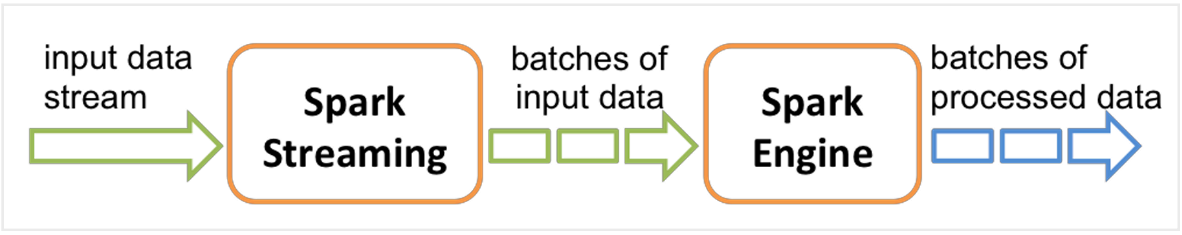
Benefits of Apache Streaming:

- Reduce latency between an event occurring and taking an action driven by it (whether automatically or via analytics presented to humans)
- Avoid the very ‘spiky’ demands on CPU/memory/etc every time a batch runs; streaming allows for dividing the processing of the same data in smaller intervals; it does this by breaking the stream up into micro-batches. Windowing capabilities are also offered for processing across multiple batches.

Data sources & destinations:

Data can be:

- ingested from multiple sources (like files, sockets, etc.),
- processed using a wide variety of high-level functions (like map, reduce, join, etc.) and
- pushed to filesystems, databases, live dashboards, etc.



Processing progresses by looking at consecutive values; going back in time probably never takes place.
The processing of such data is spread across time.

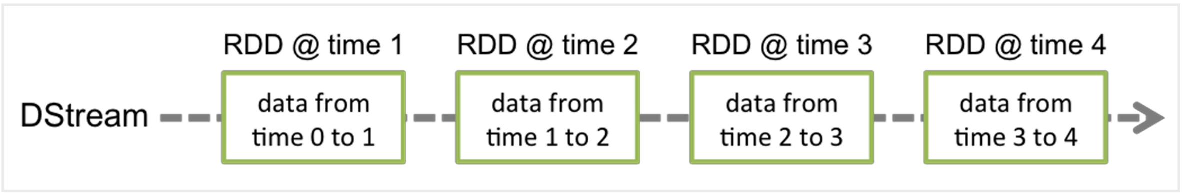
Apache Spark Streaming is an extendable platform; it can be connected to multiple sources & 'syncs' (outputs).

Its **fundamental abstraction** is the **"DStream" ("Discretized Stream"); it represents a continuous stream of data - it is represented as a sequence of RDDs.**
So we have inputs TO DStreams and outputs FROM DStreams.

A **'micro-batch'** is an RDD element produced at a particular timeframe.
Our gateway to Apache Spark Streaming is the "SparkStreamingContext"

SparkStreamingContext - DStream

"DStream" stands for "Discretized Stream". It represents a continuous stream of data. It is represented as a sequence of RDDs. Each RDD in a DStream contains data from a certain interval.



How it works
It checks the modification times of files residing in our designated directories. It captures changes based on the timeframes, and allocates them to microbatches.

```
lines = ssc.socketTextStream() -> stream of text/string data coming from a socket. The values come as line object RDDs, so we get an RDD containing line-level RDDs in it.
```

Any operation applied on a DStream translates to operations on the underlying RDDs. However, the programmer does not need to interfere; we just use the high-level methods supported.

File Streams

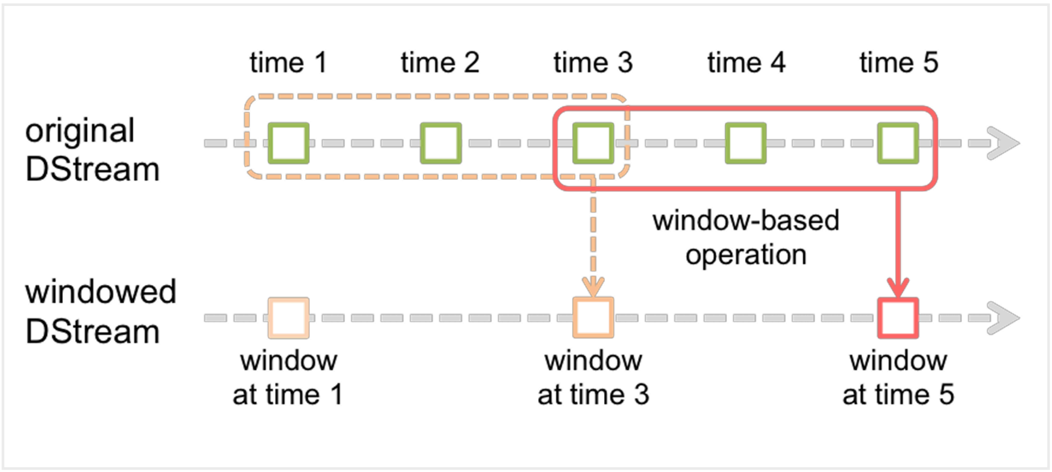
- Creation of a stream of files
  streamingContext.textFileStream(dataDirectory)
- Directory 'dataDirectory' is now monitored. It could be
  - ALL files in it need to be of the SAME DATA FORMAT
  - Files are considered part of a particular time period based on their modification times, not their creation times.

Socket Streams

A server is a script listening to a particular port and indefinitely running on localhost.
Our apache spark streaming script will be listening for data coming into that port.
We (our local machine - a client) send data to that port.

Window Operations

We can use sliding time intervals to pinpoint modifications and perform processing.



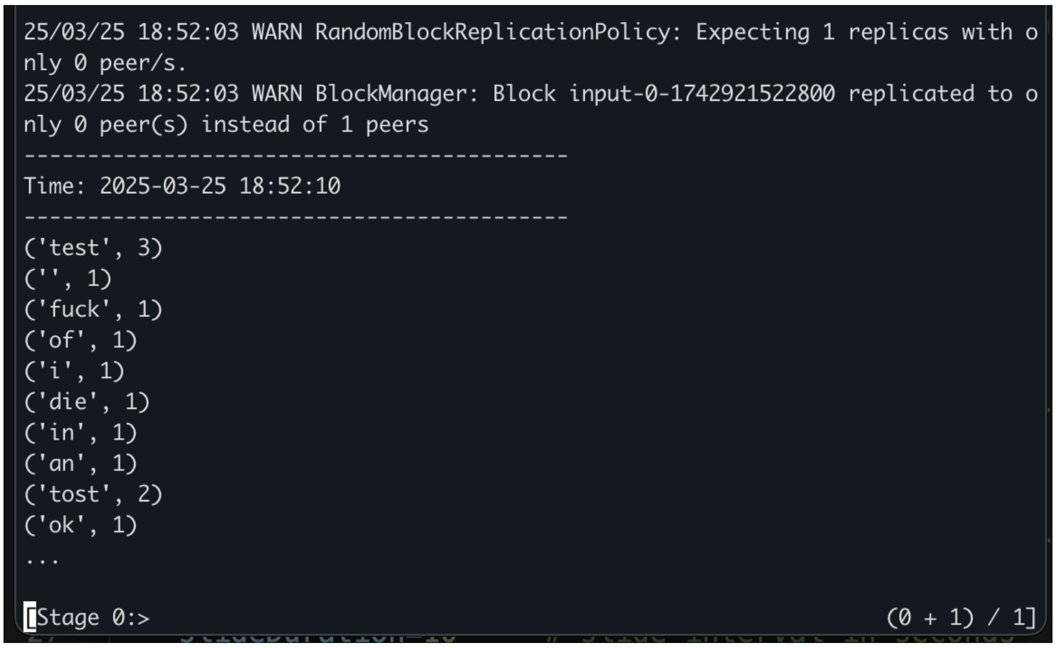
Every time the window slides over a source DStream, the source RDDs that fall within the window are combined and operated upon to produce the RDDs of the windowed DStream.

Window operations allow you to **apply transformations over a sliding window of time**, rather than just on the micro-batch that just arrived.
Real-time data often comes in bursts or spikes. If you analyze just one micro-batch (e.g., 5 seconds), the result might be too noisy or meaningless. Window operations allow you to **smooth out noise** and observe **more stable trends**.

Moreover, we might need to compute rolling metrics like ‘number of errors in the last 30 seconds’ or ‘average number of tweets per minute’ - these require us to look back at more than one micro-batch => this is exactly what window operations enable.

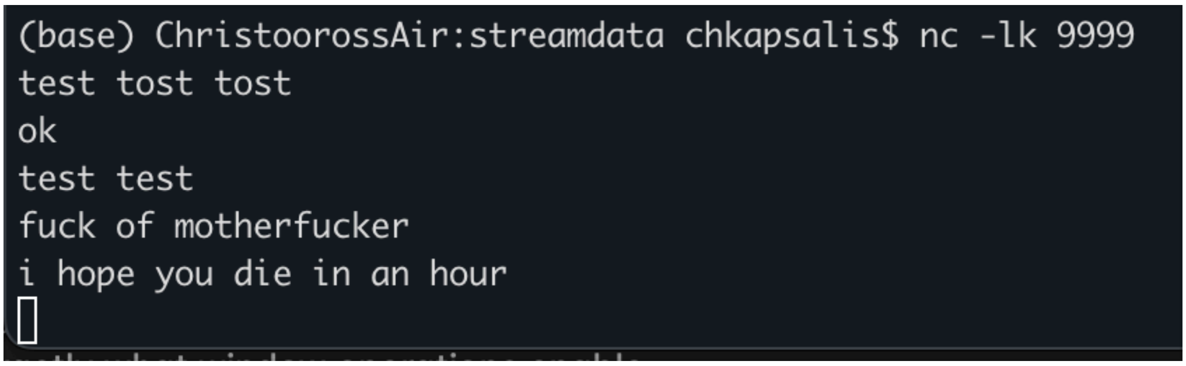
Two parameters need to be specified:

- window length: the duration of the window (e.g. 3 “micro-batches”)
- sliding interval: the interval at which the window operation is performed (e.g. ‘2’ intervals)
- These need to be multiples of the batch interval of the source DStream.



```
25/03/25 18:52:03 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with o
nly 0 peer/s.
25/03/25 18:52:03 WARN BlockManager: Block input-0-1742921522800 replicated to o
nly 0 peer(s) instead of 1 peers
-------------------------------------------
Time: 2025-03-25 18:52:10
-------------------------------------------
('test', 3)
('', 1)
('fuck', 1)
('of', 1)
('i', 1)
('die', 1)
('in', 1)
('an', 1)
('tost', 2)
('ok', 1)
...

[Stage 0:>                                                          (0 + 1) / 1]
```



```
(base) ChristoorossAir:streamdata chkapsalis$ nc -lk 9999
test tost tost
ok
test test
fuck of motherfucker
i hope you die in an hour
```