## Spark Streaming

--------------------------------------------------------------------------------------------------------------------------

A data stream is a sequence of data which is continuously arriving from the source.

Streaming divides the continuously flowing input data into discrete units for further processing.

Spark streaming was added as an extension of spark core API that provides scalable, high throughput and fault tolerant streaming process of live data streams.

Data ingestion can be done from many sources like Kafka, [Apache Flume](http://data-flair.training/blogs/introduction-apache-flume-tutorial-beginners-guide/), Amazon Kinesis or TCP sockets and processing can be done using complex algorithms that are expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases and live dashboards.

Its internal working is as follows.



Live input data streams are received and divided into batches by Spark streaming, these batches are then processed by the Spark engine to generate the final stream of results in batches.

* The data which is divided into **batches of input data** is called Dstream or Discretized Stream
* DStreams are built on Spark [RDDs](http://data-flair.training/blogs/apache-spark-rdd-tutorial/), Spark’s core data abstraction. This allows Streaming in Spark to seamlessly integrate with any other Apache Spark components like Spark MLlib and [Spark SQL](http://data-flair.training/blogs/introduction-to-apache-spark-sql-tutorial/).

#### Why Streaming in Spark?

* Batch processing systems like [Apache Hadoop](http://data-flair.training/blogs/hadoop-introduction-comprehensive-tutorial-guide-beginners/) have high latency that is not suitable for near real time processing requirements. Processing of a record is guaranteed by Storm if it hasn’t been processed, but this can lead to inconsistency as repetition of record processing might be there. The state is lost if a node running Storm goes down. In most environments, Hadoop is used for batch processing while Storm is used for stream processing that causes an increase in code size, number of bugs to fix, development effort, introduces a learning curve, and causes other issues. This creates the [difference between Big data Hadoop and Apache Spark](http://data-flair.training/blogs/introduction-spark-tutorial-quickstart/).
* Spark Streaming helps in fixing these issues and provides a scalable, efficient, resilient, and integrated (with batch processing) system. Spark has provided a unified engine that natively supports both batch and streaming workloads
* Spark’s single execution engine and unified Spark programming model for batch and streaming lead to some unique benefits over other traditional streaming systems.
* Key reason behind Spark Streaming’s rapid adoption is the unification of disparate data processing capabilities. This makes it very easy for developers to use a single framework to satisfy all the processing needs.
* Furthermore, data from streaming sources can combine with a very large range of static data sources available through Apache Spark SQL.

In spark streaming Instead of processing the streaming data one record at a time, Spark Streaming discretizes the data into tiny, sub-second micro-batches.

In other words, Spark Streaming receivers accept data in parallel and buffer it in the memory of Spark’s workers nodes.

Then the latency-optimized Spark engine runs short tasks to process the batches and output the results to other systems.

Spark tasks are assigned to the workers dynamically on the basis of data locality and available resources. This enables better load balancing and faster fault recovery.

Each batch of data is a [Resilient Distributed Dataset (RDD)](http://data-flair.training/blogs/resilient-distributed-datasets-rdd-apache-spark/) in Spark, which is the basic abstraction of a fault-tolerant dataset in Spark. This allows the streaming data to be processed using any Spark code or library.

Benefits of using Spark streaming :

------------------------------------------

1. Dynamic load balancing
2. Fast failure and straggler recovery
3. Unification of batch, streaming and interactive analytics
4. Advanced analytics like machine learning and interactive SQL
5. Performance

Spark Streaming Sources :

------------------------------

Every input DStream (except file stream) associate with a Receiver object which receives the data from a source and stores it in Spark’s memory for processing.

1. **Basic sources:** These are the sources directly available in the StreamingContext API. Examples: file systems, and socket connections
2. **Advanced sources:** Sources like Kafka, Flume, Kinesis, etc. are available through extra utility classes. These require linking against extra dependencies.

Two types of receivers base on their reliability:

-----------------------------------------------------

**Reliable Receiver:** A reliable receiver is the one that correctly sends an acknowledgment to a source when the data receives and stores in Spark with replication.

**Unreliable Receiver** – An unreliable receiver does not send an acknowledgment to a source. This we can use for sources when one does not want or need to go into the complexity of acknowledgment

### Output Operations:

* DStream’s data push out to external systems like a database or file systems using Output Operations
* Since external systems consume the transformed data as allowed by the output operations, they trigger the actual execution of all the DStream transformations.

**The following output operations**

print(), saveAsTextFiles(prefix, [suffix])”prefix-TIME\_IN\_MS[.suffix]”, saveAsObjectFiles(prefix, [suffix]), saveAsHadoopFiles(prefix, [suffix]),

foreachRDD(func)

Hence, DStreams like RDDs execute lazily by the output operations.

Specifically, the received data is processed forcefully by [RDD actions](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/) inside the DStream output operations. By default, output operations execute one-at-a-time. And they execute in the order they are defined in the Spark applications.

===============================================================