Spark

Streaming

Spark Streaming

Spark Streaming que es?

- Es un mecanismo de procesamiento en **casi tiempo real**
- Tolerante a fallos.
- Se construye encima de SparkSQL.
- Se pueden realizar las mismas operaciones que con SparkSQL.
- La herramienta se encarga de gestionar el Streaming.
- Internamente las queris se transforman en varios procesos micro-batch.

Spark Streaming

Fuentes de entrada:
Socket: Suele ser usado para pruebas
Kafka:
Ficheros:
Fuentes de salida:
Consola: Suele ser usado para pruebas
Kafka:
Ficheros:
Foreach: (recorrer cada uno de los RDD)

Spark Streaming

Modos de salida (gestión del micro-batch):

Complete mode: La tabla va siempre creciendo y mostrando todos los resultados.
Append mode: Sólo muestra los contenidos nuevos.
Update mode: Sólo muestra los contenidos que se estén incrementando.

Spark Streaming

StreamingContext
Se crea a partir de una configuración y una duración.
Igual que SparkContext en SparkSQL pero para SparkStreaming.

```
val conf = new
SparkConf().setAppName("miApp).setMaster("local[2]")
val ssc = new StreamingContext(conf, Seconds(1))
```

Esperar a que el procesamiento finalice o podamos pararlo con la función stop()

Spark Streaming

Un **Spark DStream,** que representa una secuencia de datos dividida en pequeños lotes.

Los DStreams se basan en RDD de Spark

Streaming en Spark se integre perfectamente con cualquier otro componente de Apache Spark como Spark MLlib y Spark SQL.

Spark Streaming

Fuentes de transmisión para SPARK

Cada **DStream** de entrada (excepto la secuencia de archivos) se asocia a un objeto Receiver que recibe los datos de un origen y los almacena en la memoria de Spark para su procesamiento.

Hay dos categorías de fuentes de streaming integradas:

Orígenes básicos: estos son los orígenes directamente disponibles en la API streamingContext. Ejemplos: sistemas de archivos y conexiones de socket.

Fuentes avanzadas: las fuentes como Kafka, Flume, Kinesis, etc. están disponibles a través de clases de utilidad adicionales. Estos requieren la vinculación con dependencias adicionales.

Spark Streaming

Operaciones de Spark Streaming
La transmisión por spark streaming admite dos tipos de operaciones:
i. **Operaciones de transformación en Spark**
De forma similar a los RDD de Spark, las transformaciones de Spark permiten modificar los datos de la DStream de entrada. Algunos de los más comunes son los siguientes.
map(), flatMap(),.....

ii. Operaciones de salida en Apache Spark
Los datos de DStream se insertan en sistemas externos como una base de datos o sistemas de archivos mediante operaciones de salida. Actualmente, las siguientes operaciones de salida definen como: print(), saveAsTextFiles...

Transformations on DStreams

Similar to that of RDDs, transformations allow the data from the input DStream to be modified. DStreams support many of the transformations available on normal Spark RDD's. Some of the common ones are as follows.

Transformation	Meaning
map *{func)*	Return a new DStream by passing each element of the source DStream through a function *func.*
flatMap *[func)*	Similar to map, but each input item can be mapped to 0 or more output items.
filter*func)*	Return a new DStream by selecting only the records of the source DStream on which *func* returns true.
repartii\on{numPartitions)	Changes the level of parallelism in this DStream by creating more or fewer partitions.
uri\on[otherStream)	Return a new DStream that contains the union of the elements in the source DStream and *otherDStream.*
count()	Return a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream.
reduc *e{func)*	Return a new DStream of single-element RDDs by aggregating the elements in each RDD of the source DStream using a function *func* (which takes two arguments and returns one). The function should be associative and commutative so that it can be computed in parallel.
countByValueQ	When called on a DStream of elements of type K, return a new DStream of (K, Long) pairs where the value of each key is its frequency in each RDD of the source DStream.

• reduceByKey(ft/nc, *[numTasks])*	• When called on a DStream of (K, V) pairs, return a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function. Note: By default, this uses Spark s default number of parallel tasks (2 for local mode, and in cluster mode the number is determined by the config property spark.default.parallelism) to do the grouping. You can pass an optional numTasks argument to set a different number of tasks.
• *\o\n{otherStream, [numTasks])*	• When called on two DStreams of (K, V) and (K, W) pairs, return a new DStream of (K, (V, W)) pairs with all pairs of elements for each key.
• cog ro u p(o *therStream*, *[numTasks])*	• When called on a DStream of (K, V) and (K, W) pairs, return a new DStream of (K, Seq[V], Seq[W]) tuples.
• transform(fL//7C)	• Return a new DStream by applying a RDD-to-RDD function to every RDD of the source DStream. This can be used to do arbitrary RDD operations on the DStream.
• u pd ateState By Key *[func)*	• Return a new 'state'[1] DStream where the state for each key is updated by applying the given function on the previous state of the key and the new values for the key. This can be used to maintain arbitrary state data for each key.

Output Operations on DStreams

Output operations allow DStrean's data to be pushed out to external systems like a database or a file systems. Since the output operat ons actually allow the transformed data to be consumed by external systems, they trigger the actual execution of a l the DStream transformations (similar to actions fcr RDDs). Currently, the following output ooerations are delined:

Output Operation	**Meaning**
printO	Prints the first ten elements of every batch of data in a DStream on the driver node running the streaming aoplication. This is useful for development and debugg ng. • This is called **pprintO** in the Python API.
saveAsTextFilesfpref/x, *[suffix])*	Save this DStream's cortents as text files. The file name at each batch interva is generated based on *prefix* and *suffix: "prefix-TIMEJN_MS[.suffix]*".
saveAsObjectFiles*(prefix, [suffix*D	Save this DStream's cortents as SequenceFiles of serialized Java objects. The file name at each batch interval is generated based on *prefix* and *suffix: "prefix-TIME_IN_MS[.suffix]".* • This is not available in the Python API.
saveAsHadoopFiles(p/ie//x, *[suffix]}*	Save this DStream's cortents as Hadoop files. The file name at each batch interval is generated based cn *prefix* and *suffix: "prefix-TIMEJN_MS[suffix]*[m]*.* • This is not available in the Python API.
foreachRDD(fjnc)	The most generic output operator that applies a function, *func,* to each RDD generated from the stream. This function should push the data in each RDD to an external system, such as saving the RDD to files, or writing it over the network to a database. Note that the function *func* is executed in the driver process running the streaming application, and will usually have RDD actions in it that wi 1 force the computation of the streaming RDDs.

Spark Streaming(II)

Checkpoint
La necesidad con la aplicación Spark Streaming es que debe estar **operativa las 24 horas del tiempo, los 7 días de la semana**.
Por lo tanto, el sistema también **debe ser tolerante a errores.**
Si se pierde algún dato, la recuperación debe ser rápida. La transmisión por secuencias de Spark logra **esto mediante el checkpoint**
Por lo tanto, Checkpointing **es un proceso para truncar el gráfico de linaje RDD.**

Spark Streaming (II)

hay dos tipos de puntos de control de Apache Spark:
Punto de control fiable – Se refiere a ese punto de control en el que el RDD real se guarda en un sistema de archivos distribuido confiable, por ejemplo, HDFS.

SparkContext.setCheckpointDir(directorio: String).

Punto de control local: en este punto de comprobación, en Spark Streaming o GraphX truncamos el gráfico de linaje RDD en Spark.
En este caso, el RDD se conserva en el almacenamiento local en el ejecutor.

Spark Streaming