Tema 3: Procesamiento con Apache Spark

Máster en Ciencia de Datos (Universidad de Córdoba)

UCO ONLINE

Structured Streaming

Tema 3: Procesamiento con Apache Spark

UCO ONLINE

Índice de la sección

- Introducción
- Características Principales
- Structured Streaming
- Modelo de Programación
- Ventanas de tiempo
- Marcas de agua
- Spark Streaming - DStream

Introducción

- Structured Streaming es un mecanismo de procesamiento en tiempo real y tolerante a fallos.
- Se construye encima de Spark SQL.
- Esto permitirá ingestar datos en tiempo real, a partir de ficheros, de un socket, de Apache Kafka, u otras herramientas, y realizar un procesamiento de los datos en tiempo real.
- Tipos de salidas: consola, Kafka, ficheros, o foreach.

Características Principales

- Se pueden realizar las mismas operaciones que con SparkSQL.
- La herramienta se encarga de gestionar el streaming.
- Asegura el end-to-end.
- Internamente, las queries se transforman en varios procesos micro-batch

Structured Streaming (I)

- El streaming estructurado es un motor de procesamiento de flujo escalable y tolerante a errores basado en el motor Spark SQL.
- Puede expresar su cálculo de transmisión de la misma manera que expresaría un cálculo por lotes en datos estáticos.
- El motor Spark SQL se encargará de ejecutarlo de forma incremental y continua y de actualizar el resultado final a medida que sigan llegando datos de transmisión.
- Puede usar la API Dataset/DataFrame en Scala, Java, Python o R para expresar agregaciones de transmisión, ventanas de tiempo de eventos, uniones de transmisión a lote, etc.

Structured Streaming (II)

- El cálculo se ejecuta en el mismo motor Spark SQL optimizado.
- Finalmente, el sistema asegura garantías de tolerancia a fallos de extremo a extremo a través de puntos de control y registros de escritura anticipada.
- En resumen, la transmisión estructurada proporciona un procesamiento de transmisión rápido, escalable, tolerante a fallos, de extremo a extremo sin que el usuario tenga que operar sobre la transmisión.

Structured Streaming (III)

- Internamente, de forma predeterminada, las consultas de transmisión estructurada se procesan mediante un motor de procesamiento de microlotes (micro-batchs), que procesa flujos de datos como una serie de trabajos de lotes pequeños, logrando así latencias de extremo a extremo tan bajas como 100 milisegundos y garantías de tolerancia a fallos.

Structured Streaming (IV)

- Desde Spark 2.3, se ha introducido un nuevo modo de procesamiento de baja latencia llamado Procesamiento continuo, que puede lograr latencias de extremo a extremo tan bajas como 1 milisegundo con garantías de al menos una vez.
- Sin cambiar las operaciones de Dataset/DataFrame en sus consultas, podrá elegir el modo según los requisitos de la aplicación.

Ejemplo(I)

- Ejemplo simple de una consulta de transmisión estructurada: un conteo de palabras de transmisión.
- Supongamos que desea mantener un recuento continuo de palabras de datos de texto recibidos de un servidor de datos que escucha en un socket TCP.

Ejemplo(II)

- Veamos cómo se puede expresar esto usando la transmisión estructurada.
- En cualquier caso, veamos el ejemplo paso a paso y entendamos cómo funciona. Primero, tenemos que importar las clases necesarias y crear una SparkSession local, el punto de partida de todas las funcionalidades relacionadas con Spark.

Ejemplo(III)

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split

spark = SparkSession \
    .builder \
    .appName("StructuredNetworkWordCount") \
    .getOrCreate()
```

Ejemplo(IV)

- Se crea un DataFrame de transmisión que represente los datos de texto recibidos de un servidor que escucha en localhost:9999 y transformar el DataFrame para calcular el recuento de palabras.

Ejemplo(V)

- El DataFrame de líneas representa una tabla ilimitada que contiene los datos de texto de transmisión.

```
# Create DataFrame representing the stream of input lines from connection to localhost:9999
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Split the lines into words
words = lines.select(
   explode(
       split(lines.value, " ")
   ).alias("word")
)

# Generate running word count
wordCounts = words.groupBy("word").count()
```

Ejemplo(VI)

- A continuación, se han utilizado dos funciones SQL integradas: dividir y explotar, para dividir cada línea en varias filas con una palabra cada una.
- Además, se usa la función alias para nombrar la nueva columna como "palabra".

Ejemplo(VII)

- Por último, se ha definido el DataFrame wordCounts agrupando por los valores únicos en el conjunto de datos y contándolos. Se debe tener en cuenta que se trata de un DataFrame de transmisión que representa el recuento de palabras en ejecución de la transmisión.

```
 # Start running the query that prints the running counts to the console
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

query.awaitTermination()
```

Ejemplo(VIII)

- Ahora se ha configurado la consulta en los datos de transmisión. Todo lo que queda es comenzar a recibir datos y calcular los recuentos.
- Para hacer esto, lo se configura para que imprima el conjunto completo de conteos (especificado por outputMode("complete")) en la consola cada vez que se actualicen. Y luego inicie el cálculo de transmisión usando start().

Ejemplo(IX)

- Después de ejecutar este código, el cálculo de la transmisión habrá comenzado en segundo plano. El objeto de consulta es un identificador de esa consulta de transmisión activa y hemos decidido esperar a que finalice la consulta mediante awaitTermination() para evitar que el proceso finalice mientras la consulta está activa.

```
$ nc -lk 9999
```

- Para ejecutar realmente este código de ejemplo, puede compilar el código en su propia aplicación Spark o simplemente ejecutar el ejemplo una vez que haya descargado Spark. Estamos mostrando este último. Primero deberá ejecutar Netcat (una pequeña utilidad que se encuentra en la mayoría de los sistemas similares a Unix) como un servidor de datos usando

Ejemplo(X)

- Luego, en una terminal diferente, se puede comenzar el ejemplo usando

```
$ ./bin/spark-submit examples/src/main/python/sql/streaming/structured_network_wordcount.py localhost 9999
```

Ejemplo(XI)

- Cualquier línea introducida en la terminal que ejecuta el servidor netcat, se contará e imprimirá en la pantalla cada segundo. Se verá algo como lo siguiente.

```
# TERMINAL 2: RUNNING structured_network_wordcount.py

$ ./bin/spark-submit examples/src/main/python/sql/streaming/structured_network_wordcount.py localhost 9999

-------------------------------------------
Batch: 0
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    1|
| spark|    1|
+------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    2|
| spark|    1|
|hadoop|    1|
+------+-----+
...
```

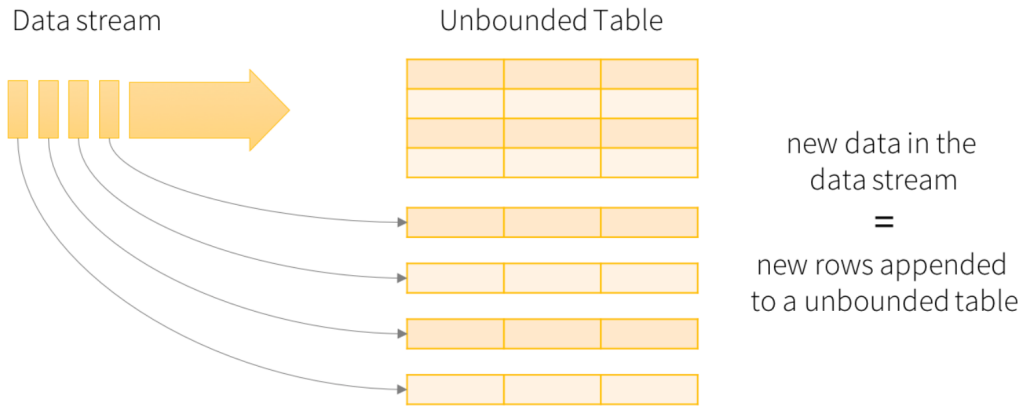
```
# TERMINAL 1:
# Running Netcat

$ nc -lk 9999
apache spark
apache hadoop

...
```

Modelo de Programación (I)

- La idea clave en la transmisión estructurada es tratar una transmisión de datos en vivo como una tabla que se agrega continuamente.
- Esto conduce a un nuevo modelo de procesamiento de flujo que es muy similar a un modelo de procesamiento por lotes.
- Expresará su cálculo de transmisión como una consulta por lotes estándar como en una tabla estática, y Spark lo ejecutará como una consulta incremental en la tabla de entrada ilimitada.

Modelo de Programación (II)

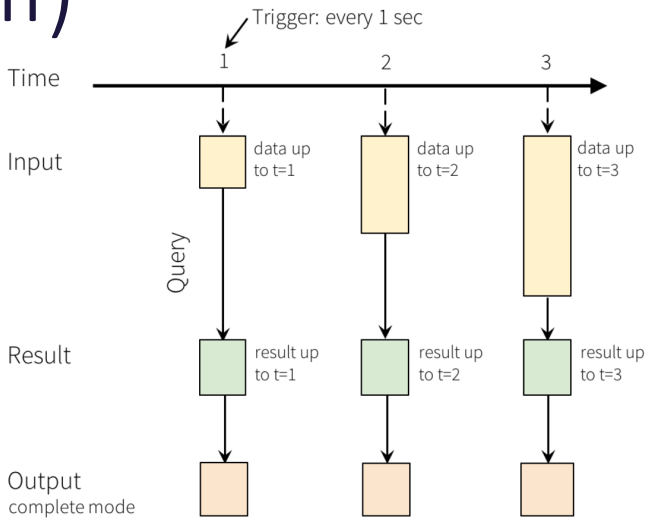
- Conceptos Básicos: Se considera el flujo de datos de entrada como la "Tabla de entrada". Cada elemento de datos que llega a la transmisión es como una nueva fila que se agrega a la tabla de entrada.



Data stream as an unbounded table

Modelo de Programación (III)

- Una consulta sobre la entrada generará la "Tabla de resultados". Cada intervalo de activación (por ejemplo, cada segundo), se agregan nuevas filas a la Tabla de entrada, que finalmente actualiza la Tabla de resultados. Siempre que se actualice la tabla de resultados, intentaremos escribir las filas de resultados modificadas en un receptor externo.



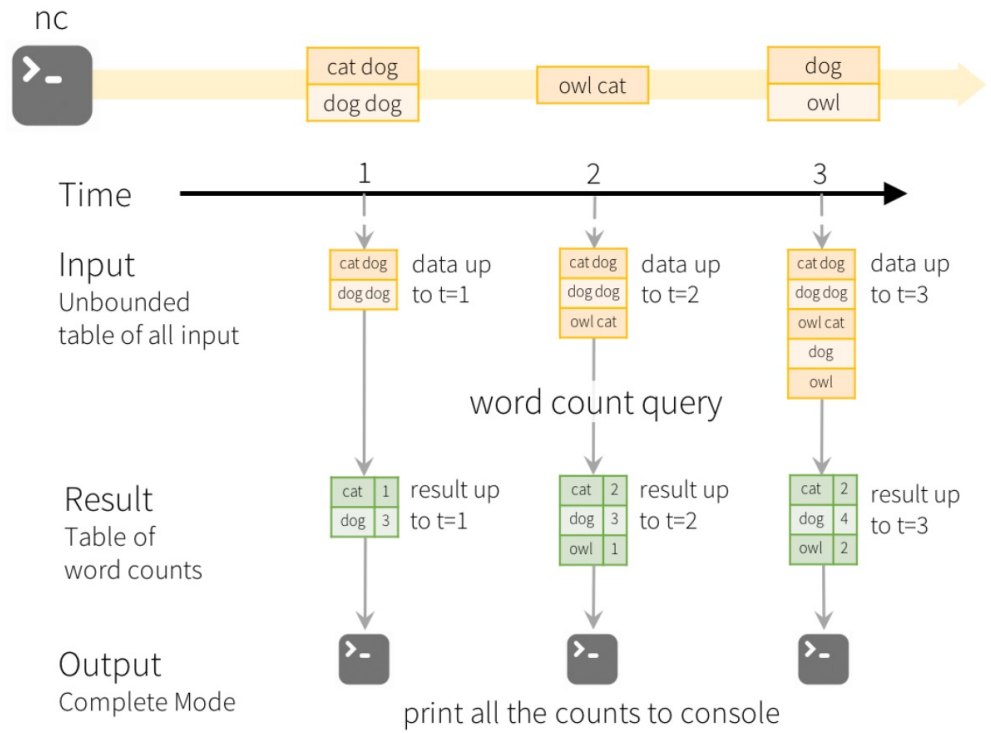
Programming Model for Structured Streaming

Modelo de Programación (IV)

- La "Salida" se define como “lo que se escribe en el almacenamiento externo”. La salida se puede definir en diferentes modos (cada modo es aplicable en ciertos tipos de consultas):
  - **Modo completo:** toda la tabla de resultados actualizada se escribirá en el almacenamiento externo. Depende del conector de almacenamiento decidir cómo manejar la escritura de toda la tabla.
  - **Modo de adición:** solo las filas nuevas añadidas en la tabla de resultados desde el último activador se escribirán en el almacenamiento externo. Esto es aplicable solo en las consultas en las que no se espera que cambien las filas existentes en la tabla de resultados.
  - **Modo de actualización:** solo las filas que se actualizaron en la tabla de resultados desde el último activador se escribirán en el almacenamiento externo (disponible desde Spark 2.1.1). Hay que tener en cuenta que esto es diferente del modo completo, en este modo solo genera las filas que han cambiado desde el último activador. Si la consulta no contiene agregaciones, será equivalente al modo adición.

Modelo de Programación (V)

- Para ilustrar el uso de este modelo, entendamos el modelo en el contexto del ejemplo rápido anterior. El DataFrame de las primeras líneas es la tabla de entrada, y el último WordCounts DataFrame es la tabla de resultados.



Model of the Quick Example

Modelo de Programación (VI)

- Hay que tener en cuenta que la consulta en las líneas de transmisión DataFrame para generar wordCounts es exactamente la misma que sería un DataFrame estático. Sin embargo, cuando se inicia esta consulta, Spark buscará continuamente nuevos datos de la conexión del socket.
- Si hay datos nuevos, Spark ejecutará una consulta "incremental" que combina los recuentos anteriores con los datos nuevos para calcular los recuentos actualizados, como se muestra a continuación.

Modelo de Programación (VII)

- La transmisión estructurada no se materializa en toda la tabla. Lee los últimos datos disponibles de la fuente de datos de transmisión, los procesa de forma incremental para actualizar el resultado y luego descarta los datos de origen.
- Solo conserva los datos de estado intermedio mínimos necesarios para actualizar el resultado (por ejemplo, recuentos intermedios en el ejemplo anterior).

Modelo de Programación (VIII)

- Este modelo es significativamente diferente a muchos otros motores de procesamiento de flujo. Muchos sistemas de transmisión requieren que el usuario mantenga las agregaciones en ejecución, por lo que debe trabajar sobre la tolerancia a fallos y la consistencia de los datos (al menos una vez, como máximo una vez o exactamente una vez).

- En este modelo, Spark es responsable de actualizar la "Tabla de resultados" cuando hay nuevos datos, liberando así a los usuarios de preocuparse a este respecto. Como ejemplo, este modelo realiza el procesamiento basado en tiempo de eventos y los datos que llegan tarde.

Modelo de Programación (IX)

- **Manejo de tiempo de evento y datos tardíos**
  - El tiempo de evento es el tiempo incrustado en los datos mismos. Para muchas aplicaciones, es posible que desee operar en este tiempo de evento.
  - Por ejemplo, si desea obtener la cantidad de eventos generados por los dispositivos IoT (Internet of Things) cada minuto, entonces probablemente desee usar la hora en que se generaron los datos (es decir, la hora del evento en los datos), en lugar de la hora en que Spark recibe a ellos.
  - Este tiempo de evento se expresa de manera muy natural en este modelo: cada evento de los dispositivos es una fila en la tabla y el tiempo de evento es un valor de columna en la fila.

Modelo de Programación (X)

- **Manejo de tiempo de evento y datos tardíos**
  - Esto permite que las agregaciones basadas en ventanas (por ejemplo, la cantidad de eventos por minuto) sean solo un tipo especial de agrupación y agregación en la columna de tiempo de evento: cada ventana de tiempo es un grupo y cada fila puede pertenecer a varias ventanas/grupos.
  - Por lo tanto, tales consultas de agregación basadas en ventanas de tiempo de eventos se pueden definir de manera consistente tanto en un conjunto de datos estáticos (por ejemplo, a partir de registros de eventos de dispositivos recopilados) como en un flujo de datos, lo que facilita mucho la vida del usuario.

Modelo de Programación (XI)

- **Manejo de tiempo de evento y datos tardíos**
  - Además, este modelo maneja de forma natural los datos que han llegado más tarde de lo esperado en función de la hora del evento.
  - Dado que Spark está actualizando la tabla de resultados, tiene control total sobre la actualización de agregados antiguos cuando hay datos atrasados, así como la limpieza de agregados antiguos para limitar el tamaño de los datos de estado intermedio.
  - Desde Spark 2.1, cuenta con soporte para marcas de agua que permite al usuario especificar el umbral de datos tardíos y permite que el motor limpie el estado anterior en consecuencia.

Modelo de Programación (XII)

- **Semántica de tolerancia a fallos**
  - Uno de los objetivos clave detrás del diseño de la transmisión estructurada fue ofrecer una semántica de extremo a extremo exactamente una vez.
  - Para lograrlo, se han diseñado las fuentes de transmisión estructurada, los sumideros y el motor de ejecución para rastrear de manera confiable el progreso exacto del procesamiento para que pueda manejar cualquier tipo de falla al reiniciar y/o reprocesar.
  - Se supone que cada fuente de transmisión tiene compensaciones (similares a las compensaciones de Kafka o números de secuencia de Kinesis) para rastrear la posición de lectura en la transmisión.

Modelo de Programación (XIII)

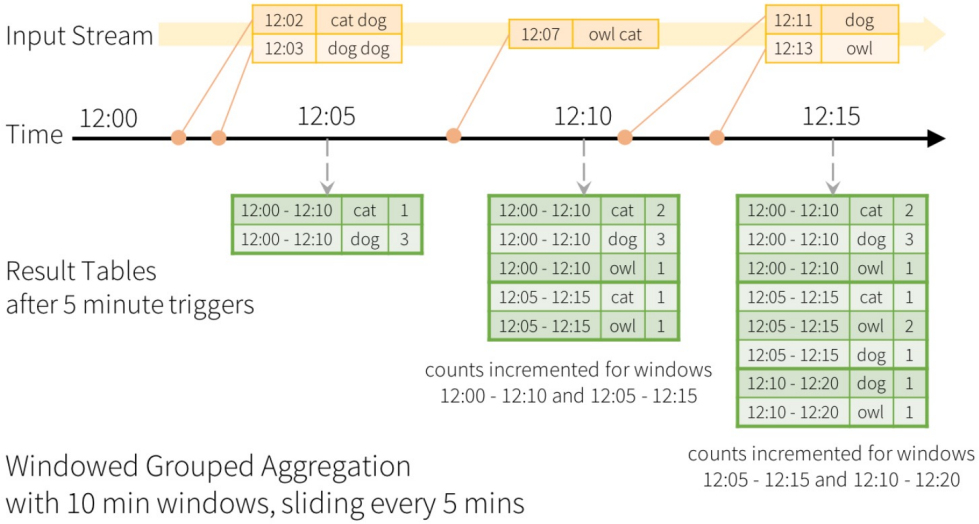
- **Semántica de tolerancia a fallos**
  - El motor utiliza registros de puntos de control y escritura anticipada para registrar el rango de compensación de los datos que se procesan en cada activador. Los sumideros de transmisión están diseñados para ser idempotentes para manejar el reprocesamiento.
  - Juntos, utilizando fuentes reproducibles y sumideros idempotentes, la transmisión estructurada puede garantizar una semántica exactamente una vez de extremo a extremo ante cualquier falla.

Ventanas de Tiempo (I)

- Las agregaciones sobre una ventana de tiempo de evento deslizante son sencillas con la transmisión estructurada y son muy similares a las agregaciones agrupadas.
- En una agregación agrupada, los valores agregados (p. ej., recuentos) se mantienen para cada valor único en la columna de agrupación especificada por el usuario.
- En el caso de agregaciones basadas en ventanas, los valores agregados se mantienen para cada ventana en la que cae la hora del evento de una fila. Entendamos esto con una ilustración.

Ventanas de Tiempo (II)

- Imagine que nuestro ejemplo rápido se modifica y la secuencia ahora contiene líneas junto con la hora en que se generó la línea.
- En lugar de ejecutar conteos de palabras, queremos contar palabras dentro de ventanas de 10 minutos, actualizando cada 5 minutos.

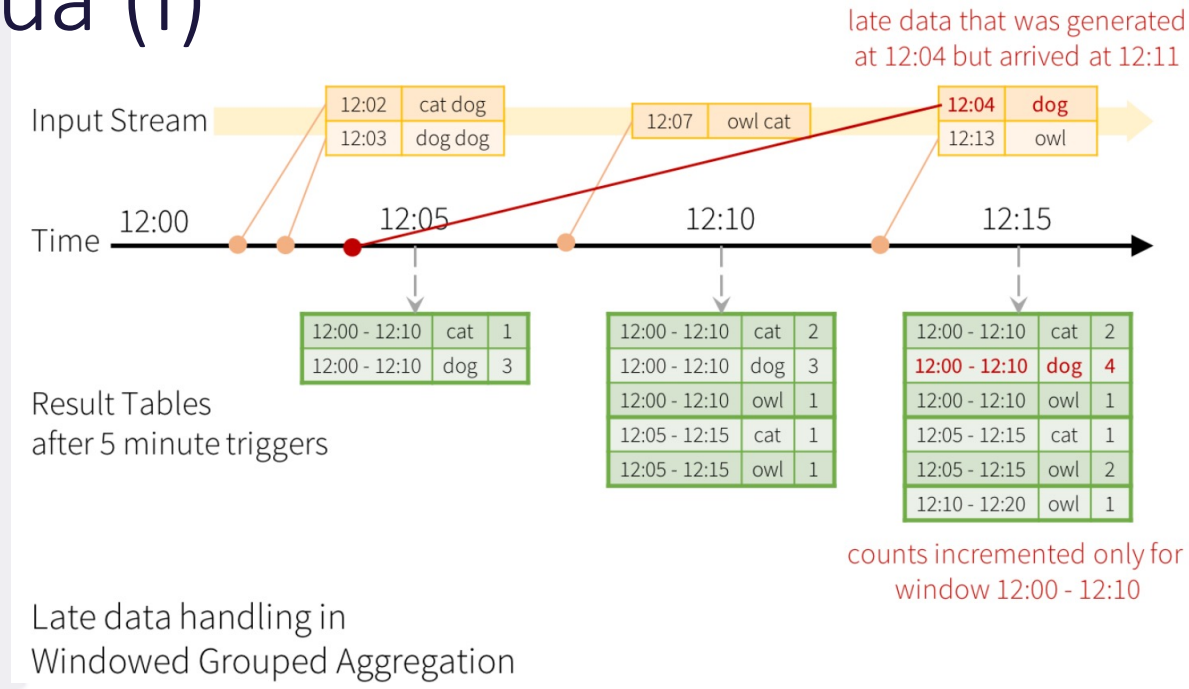


Ventanas de Tiempo (III)

- Es decir, el conteo de palabras en palabras recibidas entre ventanas de 10 minutos 12:00 - 12:10, 12:05 - 12:15, 12:10 - 12:20, etc.
- Hay que tener en cuenta que 12:00 - 12:10 significa datos que lleguen después de las 12:00 pero antes de las 12:10.
- Ahora, se considera una palabra que se recibió a las 12:07. Esta palabra debe incrementar los conteos correspondientes a dos ventanas 12:00 - 12:10 y 12:05 - 12:15.
- Por lo tanto, los recuentos se indexarán tanto por la clave de agrupación (es decir, la palabra) como por la ventana (se puede calcular a partir del tiempo del evento).

Marcas de Agua (I)

- Ahora se considera lo que sucede, si uno de los eventos llega tarde a la aplicación.



Marcas de Agua (II)

- Por ejemplo, supongamos que la aplicación podría recibir una palabra generada a las 12:04 (es decir, la hora del evento) a las 12:11.
- La aplicación debe usar la hora 12:04 en lugar de las 12:11 para actualizar los conteos anteriores para la ventana 12:00 - 12:10.
- Esto ocurre de forma natural en nuestra agrupación basada en ventanas: la transmisión estructurada puede mantener el estado intermedio para agregados parciales durante un largo período de tiempo, de modo que los datos tardíos puedan actualizar correctamente los agregados de ventanas antiguas.

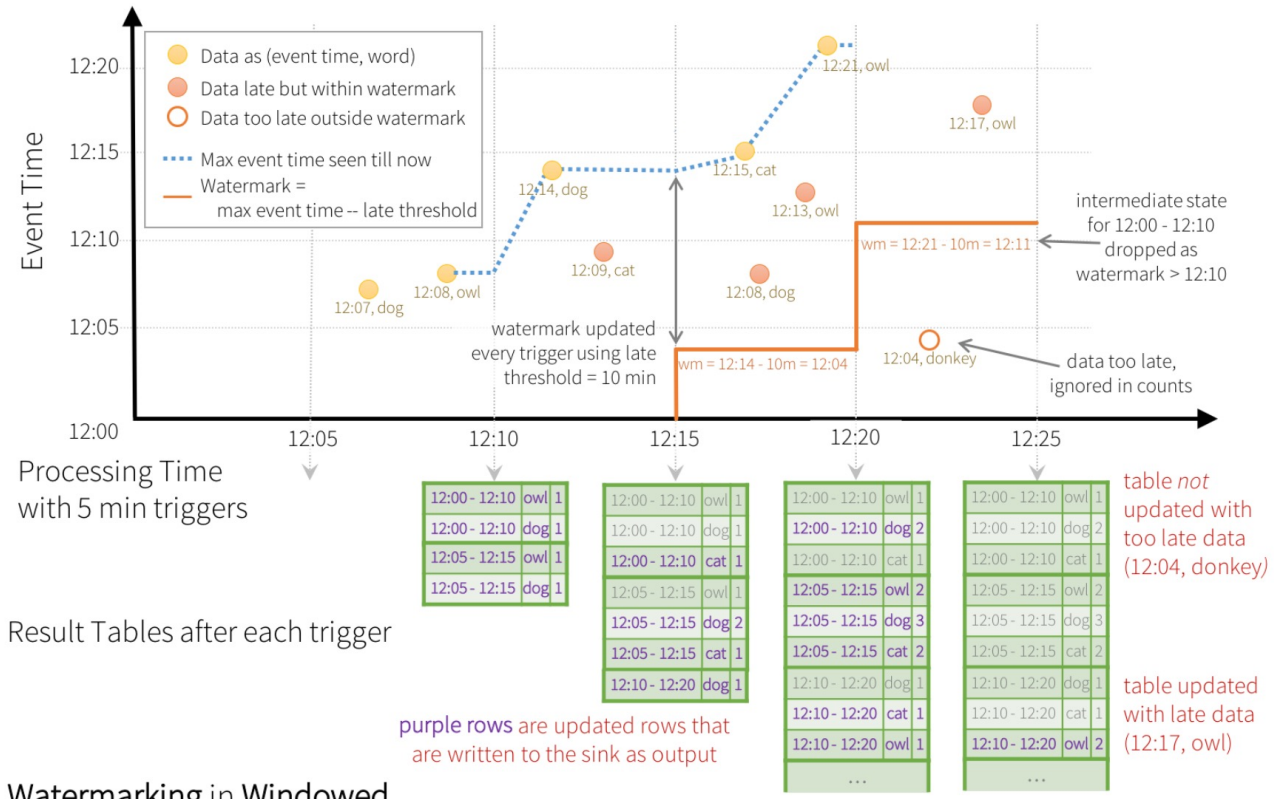
Marcas de Agua (III)

- Sin embargo, para ejecutar esta consulta durante días, es necesario que el sistema limite la cantidad de estado intermedio en memoria que acumula.
- Esto significa que el sistema necesita saber cuándo se puede eliminar un elemento agregado antiguo del estado en memoria, porque la aplicación ya no recibirá datos atrasados para ese elemento.
- Para habilitar esto, en Spark 2.1, se introdujo la marca de agua, que permite que el motor rastree automáticamente la hora del evento actual en los datos e intente limpiar el estado anterior en consecuencia.

Marcas de Agua (IV)

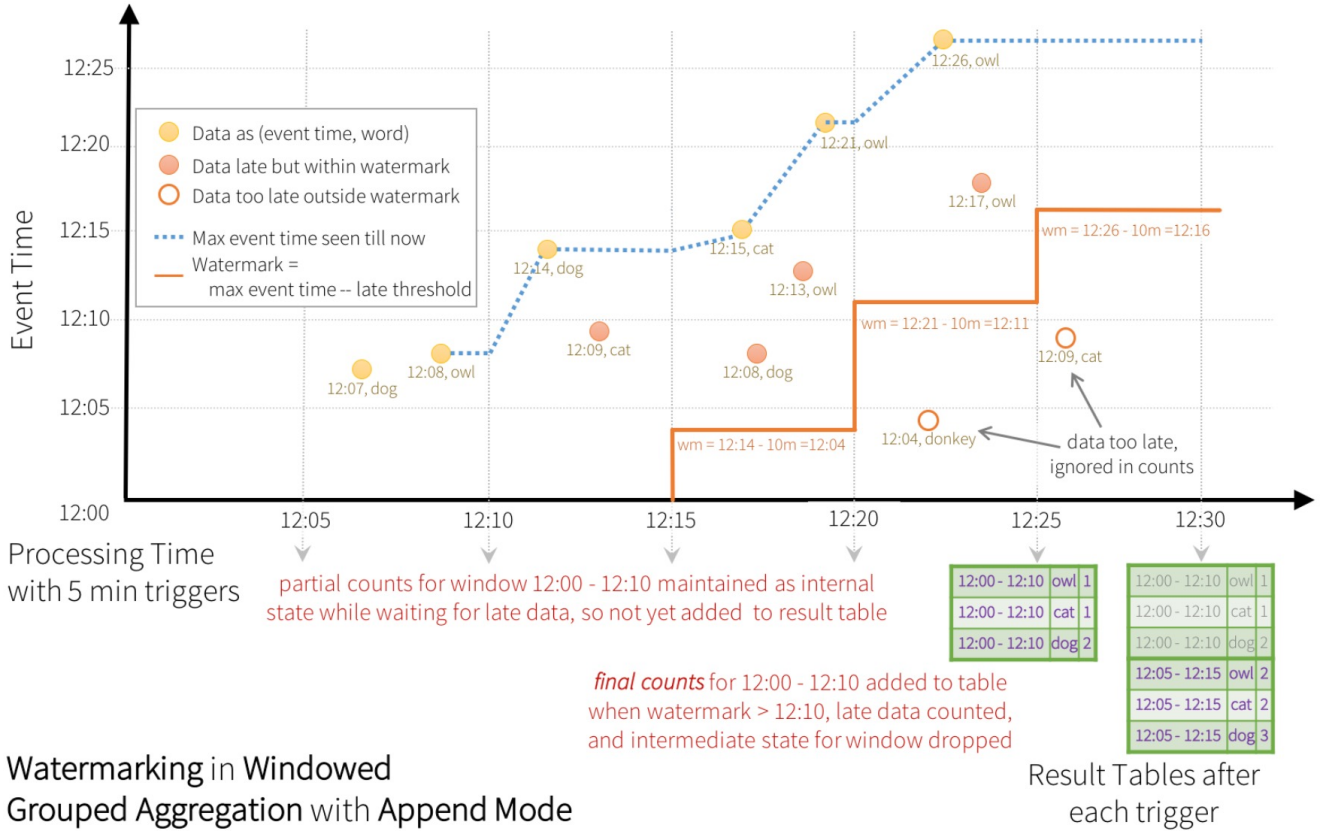
- Puede definir la marca de agua de una consulta especificando la columna de hora del evento y el umbral sobre el retraso esperado de los datos en términos de hora del evento.
- Para una ventana específica que finaliza en el tiempo T, el motor mantendrá el estado y permitirá que los datos tardíos actualicen el estado hasta (tiempo máximo del evento visto por el motor - umbral tardío > T).
- En otras palabras, los datos tardíos dentro del umbral se agregarán, pero los datos posteriores al umbral comenzarán a eliminarse.

Marcas de Agua (V)



Watermarking in **Windowed Grouped Aggregation** with **Update Mode**

Marcas de Agua (VI)



Watermarking in **Windowed Grouped Aggregation** with **Append Mode**

Operación JOIN(I)

- Consiste en tener dos fuentes de datos y hacer la unión de ambas fuentes por uno de los campos. Esta operación es similar a la que vemos en Bases de Datos.
- Structured Streaming admite la unión de un dataset/dataframe estáticos, así como con otro dataframe/dataset de transmisión.
- El resultado de la unión de transmisión se genera de forma incremental, similar a los resultados de las agregaciones de transmisión vistos.
- Se admiten diferentes tipos de combinaciones: internas, externas, semi, etc.
- Todos los tipos de unión admitidos, el resultado de la unión con un dataset o datafrma será exactamente el mismo que si fuera con un dataset o dataframe estáticos que contienen los mismos datos en la transmisión.

Operación JOIN(II)

- **Uniones estáticas de flujo:** Desde Spark 2.0, Structured Streaming admite uniones (unión interna y algún tipo de uniones externas) entre una transmisión y un DataFrame/Dataset estático.

- **Uniones de flujo a flujo:** En Spark 2.3, se tiene soporte para uniones de flujo a flujo, es decir, puede unir dos dataset o dataframe de transmisión. El desafío de generar resultados de combinación entre dos flujos de datos es que, en cualquier momento, la vista del conjunto de datos está incompleta para ambos lados de la combinación, lo que hace que sea mucho más difícil encontrar coincidencias entre las entradas. Cualquier fila recibida de un flujo de entrada puede coincidir con cualquier fila futura, aún por recibir, del otro flujo de entrada. Por lo tanto, para ambos flujos de entrada, almacenamos la entrada pasada como estado de transmisión, de modo que podamos hacer coincidir cada entrada futura con la entrada pasada y, en consecuencia, generar resultados combinados.

Operación JOIN(III)

- **Uniones internas con marca de agua opcional:** Se admiten uniones internas en cualquier tipo de columnas junto con cualquier tipo de condiciones de unión. Sin embargo, a medida que se ejecuta la transmisión, el tamaño del estado de transmisión seguirá creciendo indefinidamente, ya que todas las entradas anteriores deben guardarse, ya que cualquier entrada nueva puede coincidir con cualquier entrada del pasado. Para evitar el estado ilimitado, debe definir condiciones de combinación adicionales de modo que las entradas antiguas indefinidamente no puedan coincidir con entradas futuras y, por lo tanto, se puedan borrar del estado. En otras palabras, deberá realizar los siguientes pasos adicionales en la combinación.

1. Define los retrasos de la marca de agua en ambas entradas de modo que el motor sepa cuánto se puede estar la entrada (similar a las agregaciones de transmisión)

2. Define una restricción en el tiempo del evento en las dos entradas, de modo que el motor pueda determinar cuándo no se requerirán las filas antiguas de una entrada (es decir, no satisfará la restricción de tiempo) para las coincidencias con la otra entrada.

Operación JOIN(IV)

- **Uniones externas con marca de agua:** Si bien las restricciones de marca de agua + tiempo de evento son opcionales para las uniones internas, para las uniones externas deben especificarse. Esto se debe a que, para generar los resultados NULL en la unión externa, el motor debe saber cuándo una fila de entrada no coincidirá con nada en el futuro. Por lo tanto, las restricciones de marca de agua + tiempo de evento deben especificarse para generar resultados correctos. Una consulta con combinación externa se parecerá mucho al caso anterior, excepto que habrá un parámetro adicional que lo especificará como una combinación externa.

Spark Streaming - DStream(I)

- **Spark Streaming**: es una extensión de la API central de Spark que permite el procesamiento de diferentes flujos de datos en vivo escalable, de alto rendimiento y tolerante a fallos.
- Los datos se pueden ingerir de muchas fuentes, como Kafka, Kinesis o sockets TCP, y se pueden procesar mediante algoritmos complejos expresados con funciones de alto nivel como map, reduce, join y window.
- Finalmente, los datos procesados se pueden enviar a sistemas de archivos, bases de datos y paneles en vivo. De hecho, puede aplicar los algoritmos de aprendizaje automático y procesamiento de gráficos de Spark en flujos de datos.

Spark Streaming - DStream(II)

Spark Streaming - DStream(III)

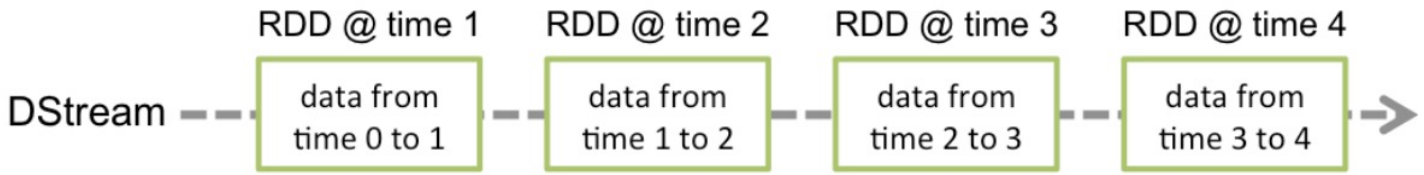
- Internamente, funciona de la siguiente manera: Spark Streaming recibe flujos de datos de entrada en vivo y divide los datos en lotes, que luego son procesados por el motor Spark para generar el flujo final de resultados en lotes.

Spark Streaming - DStream(IV)

- Spark Streaming proporciona una abstracción de alto nivel denominada **flujo discretizado o DStream**, que representa un flujo continuo de datos.
- Los DStreams se pueden crear a partir de flujos de datos de entrada de fuentes como Kafka y Kinesis, o mediante la aplicación de operaciones de alto nivel en otros DStreams. Internamente, un DStream se representa como una secuencia de RDD.

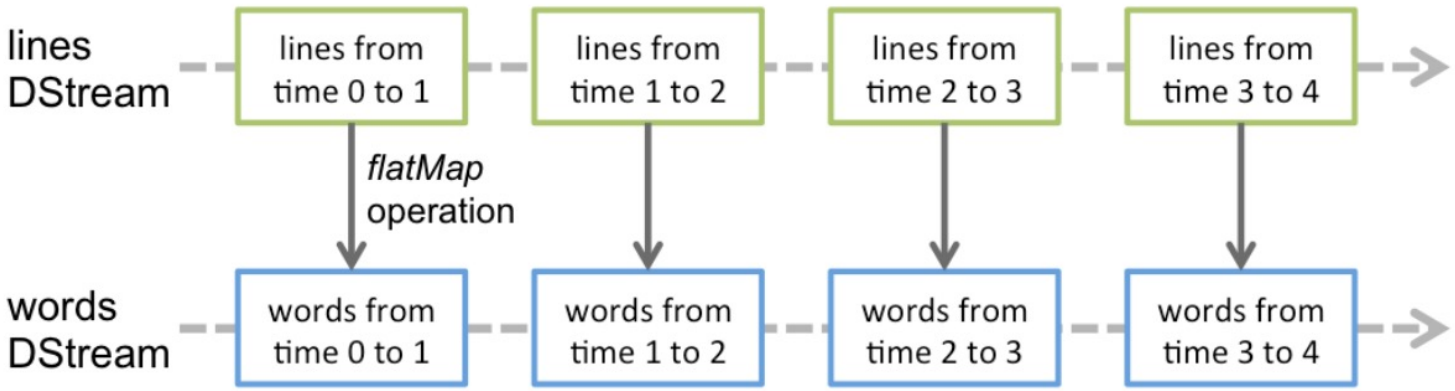
Spark Streaming - DStream(IV)

- Discretized Stream o DStream es la abstracción básica proporcionada por Spark Streaming. Representa un flujo continuo de datos, ya sea el flujo de datos de entrada recibido de la fuente o el flujo de datos procesados generados al transformar el flujo de entrada.
- Internamente, un DStream está representado por una serie continua de RDD, que es la abstracción de Spark de un conjunto de datos distribuido e inmutable. Cada RDD en un DStream contiene datos de un cierto intervalo.



Spark Streaming - DStream(V)

- Cualquier operación aplicada en un DStream se traduce en operaciones en los RDD subyacentes. Por ejemplo, en el ejemplo de convertir un flujo de líneas en palabras, la operación flatMap se aplica en cada RDD en las líneas DStream para generar los RDD de las palabras DStream.



Spark Streaming - DStream(VI)

- StreamingContext: Es el punto inicial para toda funcionalidad de este módulo.
- Se crea a partir de una configuración y una duración.
- Igual que SparkContext en SparkSQL.

¡Gracias!

UCO ONLINE