Procesamiento de Grandes Volúmenes de Datos

Conferencia 8: Procesamiento de Flujos en Tiempo Real

Deborah Famadas Rodríguez

Universidad de la Habana

3 de noviembre de 2025

El Paradigma Clásico: Procesamiento Batch

Datos Acotados (Bounded Data)

El modelo *batch* opera sobre conjuntos de datos que tienen un principio y un fin definidos.

- El conjunto de datos es **completo** y **finito**.
- Ejemplos: Un archivo CSV, una tabla de base de datos, logs de un día.
- El procesamiento se ejecuta, produce un resultado y termina.
- Es un "mundo cerrado".

El Nuevo Paradigma: Procesamiento de Flujos (Streaming)

Datos No Acotados (Unbounded Data)

El modelo *streaming* opera sobre flujos de eventos continuos y sin fin. [3]

- El conjunto de datos es **infinito** y **perpetuamente incompleto**.
- Ejemplos: Clics de usuario, telemetría de sensores, transacciones financieras.
- El procesamiento es un servicio de larga duración que **nunca termina**.
- Es un "mundo abierto".

La Perspectiva Moderna

El procesamiento *batch* es simplemente un caso especial del *streaming*: es el procesamiento de un flujo acotado.

Los Desafíos Únicos del Streaming

Procesar un flujo infinito introduce dos desafíos algorítmicos que no existen en el batch puro: **1. Gestión de Estado**

- No se puede contar un flujo infinito.
- El sistema debe **mantener y actualizar** este conteo (el estado) a medida que llegan los eventos.
- ¿Qué pasa si el servicio falla?

2. Gestión del Tiempo

- Los eventos no llegan ordenados.
- Un evento de las 10:05 puede llegar a las 10:07.
- ¿Cuándo sabemos que hemos visto *todos* los eventos de la ventana [10:00 - 10:05]?
- ¿Cómo definimos cuándo ocurrió algo?

La Dicotomía del Tiempo

La corrección de cualquier análisis de streaming depende de *qué* definición de "tiempo"se utilice.

Event Time (Tiempo de Evento)

- ¿Cuándo ocurrió el evento en la fuente?
- Es el "tiempo del mundo real".

Processing Time (Tiempo de Procesamiento)

- ¿Cuándo vio el sistema el evento?
- Es la hora del reloj local del procesador.

Desorden y Retraso

En un sistema distribuido, la latencia de red y las fallas garantizan que los eventos llegarán **con retraso y fuera de orden**.

Punto Clave

¡La brecha entre *Event Time* y *Processing Time* (skew) no es constante!

Impacto en la Corrección:

Objetivo: Contar eventos de un sensor en ventanas de 5 minutos.

- Un evento ocurre a las **10:04** (Event Time).
- Se retrasa en la red.
- El sistema lo procesa a las **10:06** (Processing Time).

Análisis por Processing Time

- El sistema ve el evento a las 10:06.
- Lo asigna a la ventana ‘[10:05 - 10:10]‘.
- **Resultado:** La ventana ‘[10:00 - 10:05]‘ es incorrecta (falta 1) y la ventana ‘[10:05 - 10:10]‘ también (sobra 1).

Análisis por Event Time

- El sistema mira el timestamp del evento (10:04).
- Lo asigna a la ventana ‘[10:00 - 10:05]‘.

El Event Time

Hemos decidido usar *Event Time* porque es el correcto.
Pero esto crea un nuevo problema:

¿Cuánto tiempo debemos esperar por los eventos de la ventana '[10:00 - 10:05]' antes de cerrarla y emitir el resultado?

- Si esperamos demasiado → Alta latencia.
- Si no esperamos lo suficiente → Alta inexactitud (perdemos datos tardíos).

La Solución: Watermarks

Necesitamos un mecanismo que nos indique el progreso en el dominio del *Event Time*. Este mecanismo es el **Watermark** (marca de agua).

Watermarks (Marcas de Agua)

Definición Conceptual

Un *Watermark* es un marcador que fluye y lleva un timestamp T. Su semántica es una declaración del sistema:
"Se asume que todos los eventos con un Event Time anterior a T ya han sido observados."

- Cuando un operador de ventana '[10:00 - 10:05]' recibe un Watermark de $T = 10:06$...
- ...el operador entiende que (presumiblemente) ya no llegarán más eventos para esa ventana.
- **Acción:** Cierra la ventana, calcula la agregación (ej. 'COUNT()') y emite el resultado.

Watermarks como Heurística

Los Watermarks no son un conocimiento perfecto, son una **heurística**.

- La heurística más común es el **retraso acotado"** (bounded-out-of-orderness).
- **Declaración de Negocio:** Asumiré que ningún evento llegará con más de 2 minutos de retraso.
- **Lógica de Generación:** El Watermark se genera como $T = (\text{max\_event\_time\_visto} - 2 \text{ minutos})$.

El Trade-off: Exactitud vs. Latencia

El diseño del Watermark (el retraso permitido") *es* el equilibrio entre exactitud y latencia.
(ej. Retraso = 1 segundo)

- Latencia muy baja pero alta inexactitud. Cualquier evento que llegue 2 segundos tarde será tardío y se descartará.

(ej. Retraso = 10 minutos)

- Alta exactitud. Captura la mayoría de los datos desordenados pero tiene latencia muy alta. El resultado de la ventana '[10:00-10:05]' no estará disponible hasta las 10:15.

Punto de Énfasis

La elección del Watermark no es técnica, es una **cuantificación del riesgo de negocio**.

¿Qué es el Windowing (Ventaneo)?

Dado que no se puede agregar un flujo infinito, el *windowing* es la técnica algorítmica principal para **dividir (acotar) el flujo** en fragmentos finitos sobre los cuales se pueden realizar cómputos.

- Es el análogo a un 'GROUP BY' en el dominio del tiempo.
- Permite responder preguntas como:
  - ¿Ventas totales *por minuto*?
  - ¿Media de tasa de error *en los últimos 5 minutos*?
  - ¿Usuarios activos *por sesión*?

Taxonomía de Ventanas: Fijas (Tumbling)

Ventanas Fijas (Tumbling Windows)

Intervalos de tamaño fijo, contiguos y que **no se solapan**.

- Cada evento pertenece a **exactamente una** ventana.
- **Caso de Uso:** Reportes periódicos. "Ventas totales cada 5 minutos".

Taxonomía de Ventanas: Deslizantes (Sliding)

Ventanas Deslizantes (Sliding/Hopping Windows)

Intervalos de tamaño fijo que **sí se solapan**.

- Se definen por un ‘tamaño‘ (duración) y un ‘deslizamiento‘ (frecuencia).
- Un evento puede pertenecer a **múltiples** ventanas.
- **Caso de Uso:** Medias móviles. ”Media de 5 minutos de la tasa de error, actualizada cada 10 segundos”.

Taxonomía de Ventanas: Sesión (Session)

Ventanas de Sesión (Session Windows)

Ventanas de tamaño **dinámico**, definidas por un *gap de inactividad*.

- Agrupan eventos que ocurren cerca en el tiempo.
- Si un nuevo evento llega dentro del *gap* (ej. 30 min), la sesión se extiende.
- Si llega *después* del *gap*, se crea una nueva sesión.
- **Caso de Uso:** Análisis de comportamiento. Agrupar clics de un usuario en una sesión de navegación.

El Ciclo de Vida de la Ventana (Modelo Dataflow)

Separación de Conceptos

- **1. Windowing (Dónde):** ¿En qué ventana de *Event Time* se agrupan los datos?
- **2. Watermark (Cuándo está Completo):** ¿Cuándo creemos que han llegado todos los datos de la ventana?
- **3. Triggers (Cuándo Materializar):** ¿Cuándo disparamos el cómputo?
  - *Trigger Temprano:* "Dame un resultado especulativo **ahora**, antes del Watermark".
  - *Trigger Tardío:* "Si llega un dato **después** del Watermark, actualiza el resultado".
- **4. Allowed Lateness (Cuánto Esperar):** ¿Cuánto tiempo *mantenemos el estado* de la ventana después del Watermark, por si acaso llegan datos tardíos?

El Desafío Más Complejo: Procesamiento Estatal

¿Qué es el Procesamiento Estatal (Stateful)?

Es cualquier operación que requiere **memoria** de eventos pasados para procesar el evento actual.

- **Stateless (Sin estado):** ‘evento.monto mayor 1000‘ (mira un solo evento).
- **Stateful (Con estado):** ‘SUM(evento.monto)‘ (necesita recordar el ‘SUM‘ anterior).

Casi todas las operaciones interesantes son estatales:

- Agregaciones (COUNT, SUM, AVG), Joins, Detección de patrones (necesita recordar el estado de la secuencia)

Modelo Lógico: Keyed State

¿Cómo escalar el estado? No puede ser global.

Keyed State (Estado Claveado)

El flujo de datos se particiona lógicamente por una **clave** (análogo a un 'GROUP BY').

- 'keyBy(userID)'
- Cada tarea de procesamiento paralela es responsable de un subconjunto de claves (ej. un *worker* maneja a los usuarios A-M, otro a N-Z).
- El estado (ej. çonteo de clics") se almacena **localmente** para esas claves.

Modelo Físico: State Backends

¿Dónde se almacena físicamente el estado?

El **State Backend** es el componente de almacenamiento que implementa la lógica del *Keyed State*.

Opciones comunes:

- **En Memoria (Heap):**
  - **Pro:** Acceso de latencia ultra baja.
  - **Contra:** Limitado por la RAM.
- **En Disco Local (ej. RocksDB):**
  - **Pro:** Permite estados que exceden la RAM (Terabytes).
  - **Contra:** Mayor latencia (acceso a disco).

Tolerancia a Fallos del Estado: Checkpointing

Problema: Si un *worker* que mantiene el estado (ej. conteos de la ventana) falla, ¡ese estado se pierde!

Solución: Checkpointing (Puntos de Control)

Un *Checkpoint* es una **"instantánea consistente"** de todo el estado distribuido de la aplicación, tomada en un punto lógico en el tiempo.

Mecánica (Algoritmo de Barrera):

1. El coordinador inyecta **barreras** especiales en el flujo.
2. Cuando un operador recibe la barrera, **toma una instantánea** de su estado local (ej. 'count=42').
3. Persiste esa instantánea en almacenamiento duradero (ej. S3, HDFS).
4. Propaga la barrera.

Recuperación mediante Checkpointing

Escenario de Fallo: Un nodo (Worker 2) falla.
Proceso de Recuperación:

1. El sistema detiene el pipeline.
2. Reinicia los operadores en nodos sanos.
3. **Cada operador recarga su estado** desde el *último checkpoint exitoso* en S3.
4. Las fuentes (ej. Kafka) se **rebobinan** a las posiciones (offsets) exactas guardadas en ese checkpoint.

Resultado

El sistema se restaura a un estado globalmente consistente y reanuda el procesamiento.

Gestión del Ciclo de Vida del Estado: TTL

Problema: En un flujo infinito, el estado con clave (ej. "última visita del usuario") puede crecer indefinidamente, incluso si el usuario nunca regresa.

"Fuga de Estado" (State Leak)

Si el estado crece sin control, eventualmente agota la memoria o el disco.

Solución: State TTL (Time-To-Live)

Permite configurar que el estado **expire automáticamente** después de un período de inactividad (ej. ."eliminar el estado de este usuario si no se actualiza en 30 días").

Usos:

- **Gestión de Recursos:** Limita el crecimiento infinito del estado.
- **Cumplimiento Normativo:** Cumple con regulaciones (ej. GDPR) que exigen la eliminación de datos.

Limpieza de Estado: Compactación

¿Cómo se elimina físicamente el estado expirado por TTL?

- **Limpieza "Perezosa" (Lazy):**
  - El estado se marca como expirado.
  - Se elimina físicamente la próxima vez que se *accede* a él (y se devuelve 'null').
- **Limpieza en Çompactación" (Compaction):**
  - Para *state backends* en disco (como RocksDB).
  - Un proceso de fondo escanea los archivos de estado.
  - Durante este proceso, se aplican filtros que descartan (limpian) los datos cuyo TTL ha expirado.
  - No es instantáneo, pero es eficiente y no bloquea el procesamiento principal.

Dualidad Flujo-Tabla (Stream-Table Duality)

Concepto fundamental (central en Kafka Streams) que unifica el streaming con las bases de datos.

Flujo como Tabla

- Un flujo puede ser visto como el **historial de cambios (changelog)** de una tabla.
- Si se reproduce el flujo de actualizaciones de ciudad de usuario desde el principio, se puede *reconstruir* la tabla que muestra la ciudad actual de cada usuario.

Tabla como Flujo

- Una tabla puede ser vista como una **instantánea (snapshot)**, en un punto en el tiempo, del *último valor* para cada clave en un arroyo.

Abstracciones: KStream vs. KTable

Kafka Streams expone esta dualidad con dos abstracciones:

KStream (Flujo)

- Abstracción de un **flujo de registros**.
- Cada registro es un **hecho** inmutable e independiente (ej. un clic, una vista, una transacción).
- "Llegó un clic", "Llegó otro clic".

KTable (Tabla)

- Abstracción de un **flujo de changelog**.
- Representa el **estado actual** para una clave.
- Un nuevo registro con una clave existente **reemplaza (actualiza)** el valor anterior para esa clave.
- El conteo de clics ES 5, El conteo de clics AHORA ES 6.

¿Preguntas?