Análisis preliminar de marcos tecnológicos en data stream

Fernando Puentes, María Dolores Pérez-Godoy, Pedro González, María José Del Jesus
Departamento de Informática
Universidad de Jaén
{fpuentes; lperez; pglez; mjjesus}@ujaen.es

***Abstract*—El análisis de datos en tiempo real está adquiriendo cada vez más importancia para ayudar en la toma de decisiones. Existen herramientas de código abierto y propietarias para el análisis de flujos continuos de datos. En este trabajo se analizan herramientas tanto para la recolección de datos de diversas fuentes como para el procesamiento de los mismos, presentando sus características más importantes con el objetivo de ayudar en la toma de decisión respecto al marco tecnológico en el desarrollo de métodos de minería de datos para *data stream*.**

Keywords*— *Streaming; Big Data; Minería de datos; Procesamiento de data stream; Recolección de datos;

I. Introducción

La minería de datos [1] se ha centrado tradicionalmente en analizar conjuntos de datos almacenados y estáticos en ambientes de procesamiento por lotes (modo *batch*). Sin embargo, cada día existen más fuentes que generan enormes cantidades de datos de forma continua, muchas veces asociados a problemas de *big data*, siendo necesario su procesamiento conforme llegan al sistema sin tener que almacenarlos (modo *streaming* o de flujo de datos).

Cada fuente va a generar un flujo de datos (*stream*) que tiene las siguientes características [2]:

- Los datos fluyen continuamente. Siempre se están generando datos, normalmente a alta velocidad.
- La distribución de los datos puede cambiar con el tiempo (cambio de concepto [3]). Debemos detectar estos cambios para poder responder a ellos lo antes posible.
- La velocidad con la que se generan los datos puede variar con el tiempo, por lo que unas veces llegarán más datos y otras menos.
- El tamaño del *stream* es teóricamente infinito, por lo que no puede ser totalmente almacenado en memoria para su procesamiento.

En los últimos tiempos, han ido apareciendo herramientas para facilitar el tratamiento de los *streams* de manera distribuida, tanto de código abierto como propietarios. La Fig. 1 muestra el esquema general de las etapas necesarias para el análisis de un *stream*. El proceso puede dividirse en 3 etapas [4]: recolección y pre-procesamiento; procesamiento; y análisis y evaluación.

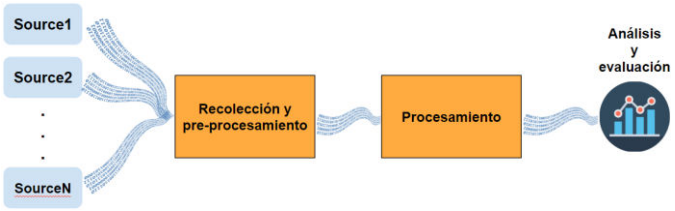

Fig. 1. Esquema general análisis en stream

En la primera etapa se utilizará una herramienta que sea capaz de realizar la recolección continua de los datos generados por una o varias fuentes. También debemos asegurarnos de que los datos estén en el formato correcto antes de pasar a la siguiente etapa, por lo que será necesario llevar a cabo un pre-procesamiento, utilizando técnicas de reducción de la dimensionalidad (reducción de características y/o de instancias) y simplificación del espacio de características [8].

Una vez que los datos han sido recolectados y pre-procesados y tienen el formato correcto, podemos pasar a la segunda etapa, el procesamiento. En esta etapa se utiliza un algoritmo de minería de datos para analizar el *stream* y llevar a cabo la extracción de conocimiento. Para mejorar el rendimiento se puede elegir una herramienta que permita realizar un procesamiento distribuido y que se adapte mejor a los requisitos y necesidades del problema a resolver.

La tercera y última etapa (cuya realización es opcional) se encarga de la visualización de los resultados obtenidos en la etapa anterior mediante alguna herramienta de visualización en tiempo real.

Aunque existen herramientas propietarias y de código abierto, en este trabajo nos centraremos fundamentalmente en las de código abierto, puesto que permiten poner el conocimiento obtenido al alcance de toda la comunidad investigadora y son cada vez más utilizadas [7]. En particular, en este trabajo se analizan herramientas que desarrollan las 2 primeras etapas, aportando información que facilite la elección de la herramienta a utilizar en función del objetivo a alcanzar.

El trabajo está organizado de la siguiente forma. En la sección 2, se detalla la etapa de recopilación y pre-procesamiento de datos y se revisan las herramientas más destacadas. En la sección 3, se introduce la etapa de procesamiento y se analizan las herramientas de esta categoría. En la sección 4, se reflejan las conclusiones obtenidas.

II. Etapa de recolección y pre-procesamiento

Antes de procesar los datos suele ser necesario recopilarlos, ya que es habitual que sean generados por diferentes fuentes simultáneas, y es necesario unificarlos para disponer de un único canal de entrada al procesamiento. Además hay que llevar a cabo un pre-procesamiento de los datos, que permita que la etapa de procesamiento reciba los datos de forma adecuada. El objetivo de esta sección es analizar y comparar diferentes herramientas para realizar estas tareas, destacando sus características principales.

A. *Apache Kafka*

Apache Kafka[1] es una plataforma de *streaming* distribuida que permite recopilar datos de diversas fuentes mediante una metodología de publicación/subscripción (el productor publica los datos y el consumidor se suscribe para recibirlos), similar a una cola de mensajes. Así, se deben definir en el clúster de Kafka un conjunto de *topics,* en los que el productor publica los mensajes, para que después un consumidor pueda recoger los datos que se van almacenando en los *topics* que se seleccionen. Estos datos se mantienen durante un tiempo (denominado periodo de retención), siendo eliminados cuando finaliza el mismo.

Los datos que se almacenan en los *topics* son pares clave-valor en formato texto, en los que la clave suele ser algún tipo de marca de tiempo (*timestamp*) y el valor el dato correspondiente. Para llevar a cabo un pre-procesamiento hay 2 opciones: hacer el pre-procesamiento en el productor o construir un consumidor-productor. La primera opción consistiría en un programa que transforme los datos antes de enviarlos y la segunda sería una aplicación que escuche un *topic*, capture todo lo que llegue a ese *topic* para transformarlo y envíe por último el resultado a otro *topic* diferente.

Las características ms importantes de Kafka se muestran en la TABLA I. Algunos puntos a destacar son:

- Tiene compatibilidad con todas las herramientas de procesamiento analizadas en este trabajo.
- Es necesario implementar un programa productor y otro consumidor, en alguno de los lenguajes soportados.
- Se dispone de una extensa documentación para desarrolladores y muchos problemas resueltos.
- No tiene una utilidad nativa para monitorizar, pero hay alguna aplicación de terceros que si lo permite, como Burrow de LinkedIn.
- Kafka es la plataforma más utilizada, ya que permite replicar y pre-procesar los datos. Además, su política de publicación/subscripción permite tener varios almacenamientos diferentes, por lo que al procesar los datos podemos elegir de qué *topic* obtenerlos, e incluso tener varias aplicaciones obteniendo datos del mismo sistema que ejecuta Kafka.

[1] https://kafka.apache.org

B. *Apache Flume*

Apache Flume[2] es un servicio distribuido que permite recolectar, agregar y mover eficientemente grandes cantidades de *logs*. Tiene una arquitectura simple y flexible basada en flujos de datos continuos. Se basa en un agente compuesto por 3 componentes: una fuente, un canal y un destino.

La fuente consume los datos de una fuente externa, como un servidor web. Esta fuente envía datos a Flume en un formato conocido por la fuente. Cuando la fuente recibe un evento, éste es almacenado en uno o más canales. Un canal es un almacén pasivo que almacena eventos hasta que son consumidos por el destino. Flume dispone de diferentes tipos de canales, dependiendo del tipo de almacenamiento que se utilice. Finalmente, el destino de los datos es el responsable de eliminar un evento del canal y ponerlo en el sistema de ficheros destino. La fuente y el destino se ejecutan asíncronamente con los eventos organizados en el canal. Hasta que un dato no se almacena en el sistema de ficheros destino, no se elimina del canal.

Las características generales más importantes de Flume se muestran en la TABLA I. Algunos puntos a destacar son:

- Se suele utilizar con Kafka (Flafka) como canal, de forma que ya no habría que implementar el productor y el receptor.
- Flume no permite replicar los datos, por lo que se suele utilizar para transportar grandes cantidades de datos de un sistema a otro.

C. *Apache Nifi*

Apache Nifi[3] es un software que permite migrar los datos de un sistema de ficheros a otro. Los datos pueden ser procesados mediante una serie de operaciones para transformarlos antes de ser enviados al destino. Cuenta con una interfaz de usuario para el navegador web, que facilita la creación de flujos y la configuración de cada una de las acciones. Nifi dispone de distintos componentes: *processor*, *reportingTask*, *controllerService*, *flowFilePrioritizer* y *authorityProvider*.

El *processor* es el componente más utilizado. Permite crear, eliminar, modificar o inspeccionar datos (denominados *flowFiles*), que incluyen un contenido y una serie de atributos que actúan como metadatos. Podemos hacer pasar los datos por diferentes *processors* para pre-procesar los datos y finalmente enviarlos al destino.

Las características ms importantes de Nifi se muestran en la TABLA I. Algunos puntos a destacar son:

- No necesita programar nada, ya que todo se configura en la interfaz.
- Permite realizar transformaciones sobre los datos, como conversión de datos/formatos, delegación de funcionalidades y operaciones de unión y difusión de los datos.

[2] https://flume.apache.org
[3] https://nifi.apache.org

- Se suele utilizar con Kafka, de forma que ya no habría que implementar el productor y el receptor.
- Permite monitorizar el estado del flujo en tiempo real, identificando posibles errores.
- Permite el envío de datos a múltiples destinos a la vez.
- Nifi no permite replicar los datos pero tiene una interfaz gráfica, por lo que se suele utilizar cuando se necesita mover datos de un sitio a otro y gestionar los *streams* de datos.

D. *Otras herramientas de recolección*

Aunque las herramientas de recolección descritas hasta ahora son de código abierto, existen otras herramientas propietarias, como Alooma o Attunity Replicate.

Alooma permite tener visibilidad y control sobre los datos, debido a su interfaz gráfica. Alooma recopila, en tiempo real, los datos de varias fuentes de datos y las unifica para aportarlas al sistema de procesamiento.

Attunity Replicate permite a las organizaciones acelerar la replicación de datos, la recolección y el *streaming* a través de diferentes bases de datos heterogéneas, almacenes de datos y plataformas de *big data*. Attunity Replicate mueve fácilmente los datos, de forma segura y eficiente.

TABLA I. Características de las herramientas de recolección

Características	**Herramienta**		
	Kafka	***Flume***	***Nifi***
Última versión	1.1.0	1.8.0	1.6.0
Fecha última modificación	28/03/2018	04/05/2018	08/04/2018
Pre-procesamiento	Si	Si	Si
Garantía de entrega	Al menos una	Al menos una	Al menos una
Tolerante a fallos	Si	Si	Si (solo con "*FileChannel*")
Lenguajes de programación	Java	Java	Java
Tiene una utilidad de monitorización	No[a]	No	Si
Plataformas	Windows Linux	Windows Linux	Windows Linux Mac OS
Documentación	Extensa	Extensa	Extensa
Replicación de datos	Si	No	No

[a] Ver información en el punto II.A

III. Etapa de Procesamiento

Una vez que se han recopilado y pre-procesado los datos, están listos para ser procesados con algún algoritmo de *streaming*. En esta sección vamos a analizar y comparar diferentes herramientas de procesamiento, destacando sus características principales. En [5] y [6] se pueden encontrar comparativas de rendimiento entre algunas de las herramientas más conocidas, como Spark, Storm y Flink.

A la hora de procesar los datos, hay que elegir de qué forma vamos a tratar los datos, utilizando alguno de los 3 tipos de garantías de procesamiento disponibles:

- Como máximo una vez: Los datos pueden ser procesados una vez o ninguna.
- Al menos una vez: Los datos pueden ser procesados 1 o más veces. Nunca tendremos datos sin procesar.
- Exactamente una vez: Los datos serán procesados una sola vez. No hay ni duplicados, ni datos no procesados.

A. *Apache Spark Streaming*

Apache Spark Streaming[4] es una extensión del núcleo (*core*) de Spark que permite procesamiento escalable, de alto rendimiento y tolerante a fallos de *streams* de datos. Los datos llegan a través de uno o varios *streams* de entrada que pueden ser procesados usando algoritmos complejos expresados con funciones a alto nivel como *map*, *reduce*, *join* y *window*. Finalmente, los datos procesados pueden ser transferidos a un sistema de ficheros o una base de datos.

Spark Streaming trabaja de la siguiente forma. Recibe un *stream* de entrada de datos y lo divide en *batches*, todos del mismo tiempo (todos los datos que lleguen en un determinado tiempo se tendrán en cuenta). Después, cada *batch* es procesado por el *Spark Engine* mediante algún algoritmo, generando a la salida un *batch* por cada *batch* de entrada.

Las principales características se muestran en la TABLA II. Algunos puntos a destacar son:

- Puede analizar los datos en modo *batch* o en modo *streaming* (una adaptación del modo *batch* en el que el *stream* se divide en *batches*).
- Permite la utilización de ventanas temporales. Éstas tienen un tamaño de ventana y un desplazamiento que debe ser múltiplo del intervalo de *batch*.
- La latencia es mayor que en otras herramientas, ya que no realiza un procesamiento continuo de los datos, sino que tiene que esperar siempre a que se cumpla el tiempo de un *batch*.
- Spark Streaming es útil en aquellos casos en los que necesitamos procesar datos por lotes en modo *batch* o en modo *streaming*. Además, la existencia de librerías de algoritmos, su extensa documentación y su activa comunidad pueden ser factores por los que elegir esta herramienta.

B. *Apache Spark Structured Streaming*

Apache Spark Structured Streaming[5] es un motor de procesamiento de *stream* escalable y tolerante a fallos construido sobre el motor de *Spark SQL*, lo que permite procesar un *stream* de manera similar a como se hace en el modo *batch* sobre datos estáticos. Structured Streaming permite hacer consultas que se resuelven incrementalmente,

[4] https://spark.apache.org
[5] https://spark.apache.org/docs/2.3.1/streaming-programming-guide.html

actualizando el resultado final, que se almacena en una tabla de salida.

En el modo *streaming*, hay una tabla ilimitada (*dataset*) a la que se van añadiendo los datos conforme llegan por el *stream*. Sobre esta tabla se realizan las consultas incrementales. Estas consultas procesan los datos en *micro-batches* de 100 milisegundos, aunque en la nueva versión 2.3 de Spark se ha introducido un modo continuo, denominado "Procesamiento continuo", que permite latencias por debajo de 1 milisegundo con garantías de procesamiento de "al menos una vez", pero es una característica experimental aún en desarrollo.

Las principales características se muestran en la TABLA II. Algunos puntos a destacar son:

- Permite procesamiento en modo *batch* y en modo *streaming*.
- En el momento de actualizar la salida hay 3 modos: *complete*, *append* y *update*. Dependiendo del tipo de consulta se podrá utilizar uno u otro.
- En otros modelos, el usuario tiene que mantener y actualizar las agregaciones ejecutadas. En este modelo, Spark es el que se encarga de mantener y actualizar las tablas con los nuevos datos.
- Permite utilizar ventanas temporales igual que Spark Streaming, con la diferencia de que aquí no hay que especificar el intervalo de *batch*.
- Junto con las ventanas temporales se puede utilizar lo que se conoce como *watermark*, que establece un límite para los datos que lleguen con retraso, basándose en un campo *timestamp*. Si un dato llega con retraso, se actualizan los valores de su instante temporal correspondiente y no en el que llega el dato.
- No dispone de ninguna librería con algoritmos de minería de datos, aunque si hay algunas implementaciones en GitHub.
- Spark Structured Streaming es útil en aquellos casos relacionados con consultas a bases de datos, gracias a sus consultas incrementales sobre los datos que llegan. Además, la utilización de *watermarks* con ventanas temporales pueden ser factores por los que elegir esta herramienta.

C. Apache Storm

Apache Storm[6] es un sistema de computación distribuida en tiempo real que procesa los datos de entrada dato a dato. Storm trabaja por topologías, de forma que el flujo va a ir pasando por una serie de nodos/operaciones hasta llegar al final de la topología. En una topología Storm existen dos tipos de nodos: Spout, que regulan la entrada de datos al sistema, y Bolt, que realizarán operaciones sobre los datos.

Storm tiene un modo de funcionamiento utilizando una abstracción superior denominada Trident, lo que permite que pueda procesar los datos por lotes (*batch*). Este modo ofrece garantías de procesamiento de "exactamente una vez", aunque aumenta la latencia al tener que esperar para recopilar un conjunto de datos.

La TABLA II muestra las principales características de Storm. Algunos puntos a destacar son:

- Solo permite procesamiento en modo *streaming*, dato a dato.
- No garantiza que los datos se procesen en orden.
- Permite la utilización de ventanas temporales y, junto a estas, se pueden utilizar *watermarks*.
- No dispone actualmente de ninguna librería con algoritmos de minería de datos, aunque hay en desarrollo una denominada SAMOA[7].
- Storm es útil en aquellos casos en los que se van a repartir las tareas por nodos según una topología y queremos procesar los datos dato a dato con garantías de procesamiento de "al menos una vez".

D. Apache Flink

Apache Flink[8] es un *framework* de procesamiento para aplicaciones de *streaming* de datos distribuidas que permite procesar los datos dato a dato o en *micro-batches*. También permite el procesamiento en modo *batch*, con un conjunto de datos estático. Funciona por topologías, al igual que Storm, en las que hay 3 tipos de nodos: operadores, fuentes y destinos. Los datos entrarán por los nodos fuente al sistema, serán procesados mediante diferentes nodos operadores y el resultado será depositado en un destino.

Flink es muy potente en cuanto al uso de ventanas, ya que permite una gran variedad. Las ventanas pueden estar basadas en número de eventos o en tiempo. Dentro de una ventana también se permite distinguir entre eventos de diferentes tipos. Hay diferentes tipos de ventana ya implementadas, pero se ofrece la posibilidad de implementar una ventana personalizada si fuese necesario.

Las características principales se muestran en la TABLA II. Algunos puntos a destacar son:

- Su mayor ventaja es la variedad de ventanas, que además permiten la utilización de *watermarks* para permitir un retardo en los datos de entrada.
- Permite procesar los datos en modo *streaming* y en modo *batch*. Además, se puede elegir entre procesar los datos de uno en uno o por lotes.
- Flink es útil en casos en los que queremos procesar los datos dato a dato o por lotes con garantías de procesamiento de "exactamente una vez", en modo *batch* o en modo *streaming*. Flink es muy similar a Storm, pero proporciona funcionalidades que en Storm habría que implementar. Además, la posibilidad de utilizar ventanas temporales junto con *watermarks* o de un tamaño concreto puede ser un factor importante por el que elegir esta herramienta.

[6] http://storm.apache.org

[7] https://samoa.incubator.apache.org

[8] https://flink.apache.org

E. Apache Samza

Apache Samza[9] es un *framework* de procesamiento distribuido de *stream* que utiliza Kafka y YARN en el proceso. Samza ofrece muchas funcionalidades similares a Storm y procesa *streams* de datos mediante tareas predefinidas, que realizan alguna operación en el *stream* de datos. Una aplicación Samza es un flujo de datos que consiste en consumidores que obtienen datos que son procesados por un grafo de trabajos, donde cada trabajo contiene una o más tareas. En Samza cada trabajo es una entidad que puede ser desplegada, iniciada o parada inmediatamente.

En Samza, cada tarea contiene un almacén clave-valor usado para almacenar el estado. Los cambios en este almacén son replicados a otras máquinas del clúster para permitir a las tareas ser recuperadas rápidamente en caso de fallo.

Las características de Samza se muestran en la TABLA II. Algunos puntos a destacar son:

- Garantiza que los mensajes son procesados en orden.
- No dispone de una librería con algoritmos de minería de datos, aunque en el futuro podrá utilizar algoritmos de SAMOA.
- Utiliza procesos *single-thread*.
- Samza es útil en casos en los que necesitamos procesar los datos dato a dato, en modo *batch* o modo *streaming*. Samza se ha realizado sobre Kafka, por lo que los datos se van a almacenar en particiones y se asegura que van a ser procesados en orden.

F. Apache Apex

Apache Apex[10] es una plataforma de procesamiento basada en YARN nativo de Hadoop. Apex proporciona un API simple, que permite a los usuarios escribir código genérico y reusable. El código se coloca como está y la plataforma automáticamente maneja las cuestiones operativas, como gestión de estados, tolerancia a fallos, escalabilidad, seguridad, métricas, etc. Esto permite que los usuarios se centren en el desarrollo.

El núcleo de la plataforma de Apex es complementado por Malhar, una librería de funciones lógicas y conectores. Proporciona acceso a diferentes sistemas de ficheros, sistemas de mensajes y bases de datos.

Las características principales se muestran en la TABLA II. Algunos puntos a destacar son:

- La topología de Apex es un grafo acíclico dirigido, cuyos nodos son Operators, y los arcos son los *streams*.
- Los datos se procesan dato a dato, aunque también soporta procesamiento por lotes de un determinado tiempo.
- No dispone de una librería con algoritmos de minería de datos, aunque en el futuro podrá utilizar algoritmos de SAMOA.
- Apex es útil en casos en los que necesitamos procesar los datos dato a dato con garantías de procesamiento de "exactamente una vez" y se quiere crear y utilizar código que sea reutilizable. Además tiene compatibilidad con la librería Malhar, que puede ser un factor importante para elegir esta herramienta.

G. Apache Beam

Apache Beam[11] es un *framework* de procesamiento de *streams* que permite definir un procesamiento independientemente del sistema de procesamiento de datos (denominado *runner*) que se utilice. Permite seguir utilizando el mismo procesamiento programado, permitiendo cambiar el *runner*. Permite utilizar los siguientes *runners*: Apache Apex, Apache Flink, Apache Spark, Apache Gearpump y Google Cloud Dataflow.

Sus características se muestran en la TABLA II. Algunos puntos a destacar son:

- La garantía de procesamiento depende del *runner* que se utilice para procesar los datos, ya que cada uno ofrece unas garantías propias de la herramienta.
- Se puede utilizar como lenguaje de programación Java y Python, aunque también hay una API de Scala en un proyecto de GitHub[12].
- La latencia varía según el *runner* que se utilice.
- Beam es útil en casos en los que se quiere probar un procesamiento con diferentes *runners* para ver cual se adapta mejor.

H. Otras herramientas de procesamiento

Aunque todas las herramientas de procesamiento analizadas aquí son de código abierto, existen otras herramientas propietarias, como Amazon Kinesis o Google Cloud Dataflow. Estas herramientas ofrecen soporte, mayor seguridad y herramientas avanzadas como entornos de desarrollo e interfaces orientadas al negocio [7].

Amazon Kinesis permite procesar y analizar datos a medida que se reciben y es capaz de ofrecer una respuesta instantánea en lugar de tener que esperar a la recopilación de datos. Una vez procesados los datos, Kinesis ofrece también herramientas para visualizar los resultados mediante gráficas o tablas.

Google Cloud Dataflow es un servicio de procesamiento de datos administrado capaz de ejecutar canalizaciones, tanto en modo *batch* como en modo *streaming*. Utiliza un modelo de programación unificado e incluye SDKs para definir flujos de procesamiento de datos.

Finalmente, hay una herramienta llamada Apache Gearpump, que aún está en desarrollo (*incubating*). Gearpump es un motor de procesamiento de *streaming* basado en eventos/mensajes que está inspirado en los avances recientes en el *framework* de Akka.

[9] http://samza.apache.org
[10] https://apex.apache.org
[11] https://beam.apache.org
[12] https://github.com/spotify/scio

IV. Conclusiones

En este trabajo se han analizado diferentes herramientas tanto para recolectar datos de las fuentes que los generan como para procesarlos. Cada herramienta tiene ventajas e inconvenientes según la situación, por lo que sus características serán factores clave a la hora de seleccionar la herramienta más apropiada para nuestro objetivo.

En cuanto a recolección de datos, la herramienta más completa es apache Kafka, ya que tiene un gran rendimiento para esta tarea y permite pre-procesar los datos. También permite replicar los datos, lo que aumenta considerablemente la tolerancia a fallos. Además, Kafka puede recopilar datos para diferentes aplicaciones a la vez, ya que se pueden distinguir por un *topic* diferente. Por último, esta herramienta está siendo muy utilizada y dispone de gran cantidad de documentación y problemas resueltos para ayudar a los desarrolladores.

En cuanto a procesamiento, actualmente la mejor herramienta es Apache Spark Streaming por su gran rendimiento, a pesar de que tiene algo de latencia. Además, tiene garantías de procesamiento de "exactamente una vez" y tiene algunas librerías con algoritmos ya implementados, tanto en modo *batch* como en modo *streaming*, lo que puede ayudar a los desarrolladores a la hora de implementar nuevos algoritmos. También tiene compatibilidad con Kafka, por lo que se pueden utilizar ambos para las etapas correspondientes. Por último, esta herramienta también está siendo ampliamente utilizada y dispone de gran cantidad de documentación útil para los desarrolladores, evitando tener que invertir mucho tiempo para resolver problemas.

Agradecimientos

Este trabajo ha sido subvencionado por el Ministerio de Economía y Competitividad bajo el proyecto TIN2015-68454-R, Fondos FEDER.

Referencias

[1] J. Han, M. Kamber y J. Pei. Data Mining: Concepts and Techniques, 3rd Edition. Morgan Kaufmann, 2011

[2] J. Gama. Knowledge Discovery from Data Streams. Chapman & Hall/CRC, 2010

[3] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, K. Ghédira. Discussion and review on evolving data streams and concept drift adapting. Evolving Systems 9:1–23, 2018

[4] Ms.D.Jayanthi, Dr.G.Sumathi. A Framework for Real-time Streaming Analytics using Machine Learning Approach. In: Proceedings of National Conference on Communication and Informatics-2016

[5] J. Samosir, M. Indrawan-Santiago, P. D. Haghighi. An evaluation of data stream processing systems for data driven applications. Procedia Computer Science 9:439-449, 2016

[6] Y. Wang. Stream Processing Systems Benchmark: StreamBench. Ph. D. dissertation. Aalto University, School of Science, 2016

[7] Gartner. Market Guide for Event Stream Processing (G00332885), 2018

[8] S. Ramírez-Gallegoa, B. Krawczyk, S. García, M. Wozniak, F. Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing 239:39-57, 2017

TABLA II. Principales características de las herramientas de procesamiento

Características	**Herramientas**						
	Spark Streaming	***Spark Structured Streaming***	***Storm***	***Flink***	***Samza***	***Apex***	***Beam***
Última versión	2.3.1	2.3.1	1.2.2	1.5.0	0.14.1	3.7.0	2.4.0
Fecha última modificación	08/06/2018	08/06/2018	04/06/2018	25/05/2018	18/05/2018	27/04/2018	20/03/2018
Garantía de procesamiento	Exactamente una vez	Exactamente una vez	Al menos una vez / Exactamente una vez	Exactamente una vez	Al menos una vez	Exactamente una vez	Depende[a]
Tolerante a fallos	Si	Si	Si	Si	Si	Si	Si
Lenguajes de programación	Java Scala Python R	Java Scala Python R	Java Scala Python Ruby	Java Scala Python	Java	Java	Java Scala[a] Python
Utilidad de monitorización	Si	Si	Si	Si	Si	Si	Si
Plataformas	Windows Linux Mac OS	Windows Linux Mac OS	Windows Linux	Windows Linux Mac OS	Linux	Windows Linux Mac OS	Windows Linux Mac OS
Documentación	Muy extensa	Extensa	Extensa	Extensa	Poca	Poca	Extensa
Modificaciones durante la ejecución	No	No	Si	No	No	Si	No
Latencia	Segundos	Milisegundos	Milisegundos	Milisegundos	Milisegundos	Milisegundos	Depende[a]
Procesamiento modo batch	Si	Si	Si[a]	Si	Si	No	Si
Procesamiento modo streaming	Si	Si	Si	Si	Si	Si	Si
Librería Minería de Datos	MLLIB streamDM Amidst Toolbox	No[a]	SAMOA[a]	FlinkML SAMOA Amidst Toolbox	SAMOA[a]	SAMOA[a]	Si[a]

[a] Ver información en el punto correspondiente