Gestión de los datos y su uso: *data analytics*

PID_00247342

José López Vicario

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

Índice

Introducción 5

1. Quinta V del *big data* 6

2. *Big data analytics* 7

3. *Batch processing analytics* 10
3.1. Descripción de la metodología 10
3.2. *Machine learning* con Spark 11
3.3. Ejemplo de aplicación 12

4. *Real time analytics* 14
4.1. Descripción de la metodología 14
4.2. Arquitectura de Apache Storm 15
4.3. Algoritmos utilizados 16
4.4. Ejemplo de aplicación 17

5. Arquitectura lambda 19

Resumen 21

Bibliografía 22

Introducción

Este material se centra en la obtención de información de valor añadido a partir del análisis de los datos capturados, que en inglés se denomina *data analytics*. Como se mencionó en el material «Fundamentos del *big data*: arquitectura del sistema», cuando se hace referencia a *big data*, se habla de las 4 V: volumen, velocidad, variedad y veracidad de los datos. No obstante, en muchos entornos se añade una quinta V para referirse al *valor* que puede extraerse de los datos, que es lo que tendremos en cuenta aquí.

Este material distingue entre estrategias llevadas a cabo en tiempo real y aquellas basadas en la obtención de *analytics* mediante posprocesado. Para cada una de estas estrategias, se hace un barrido de las principales arquitecturas y herramientas utilizadas. En este barrido, se hace una revisión de los principales *insights* que pueden obtenerse del análisis de *big data* en un entorno industrial, mediante la presentación de ejemplos.

Finalmente, se presenta una arquitectura de procesado de datos híbrida, conocida como *arquitectura lambda*, la cual combina las ventajas de un mecanismo en tiempo real (respuesta con baja latencia) con las de un sistema basado en posprocesado (mejores prestaciones, ya que dispone de un conjunto mayor de datos).

1. Quinta V del *big data*

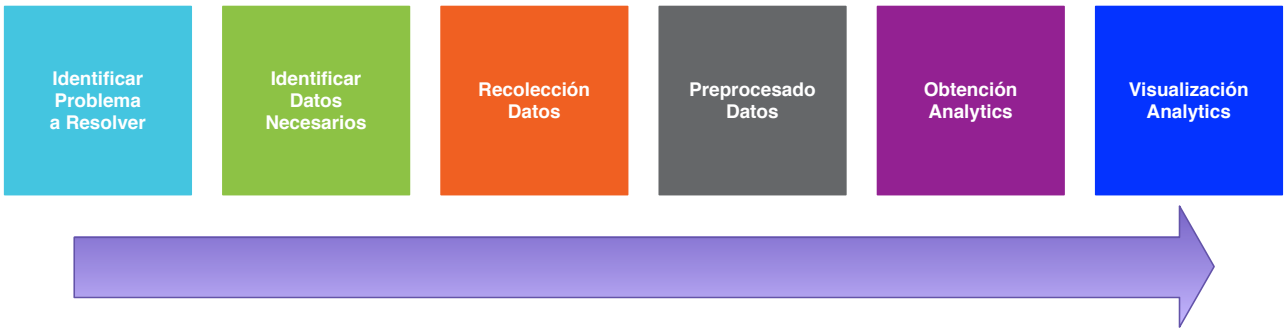
Tal y como se ha mencionado anteriormente, la quinta V de *big data* hace referencia al valor que puede obtenerse de los datos. Ciertamente, es muy común leer aseveraciones respecto a que *big data* proporciona mucho valor a las organizaciones, pero si este valor no puede cuantificarse, difícilmente resulta medible. Desde un punto de vista económico, el valor se puede medir como los beneficios adicionales que puede conseguir una organización mediante la explotación de los datos. Teniendo en cuenta que beneficios es igual a ingresos menos costes, el valor de los datos vendrá medido por los ingresos adicionales que puedan conseguirse o a la reducción de costes obtenida. Por tanto, es importante ser capaz de medir qué diferencial (en ingresos o costes) puede obtenerse gracias al uso de *big data*.

Por poner un ejemplo directamente relacionado con el entorno industrial, que utilizaremos a lo largo de este material, veamos el valor obtenido en el caso de mantenimiento predictivo. Concretamente, el valor que se obtiene al predecir que una maquinaria tiene alta probabilidad de presentar una anomalía se basa en el ahorro económico al detener esta máquina para ejecutar un mantenimiento preventivo (de manera que se modifica la planificación de la planta de forma acorde), frente a la pérdida de ingresos que se sufriría al alterar la producción repentinamente (con poco margen de maniobra), junto con el mayor coste económico por tener que reparar la maquinaria o sustituirla.

2. *Big data analytics*

Big data analytics se refiere al proceso de analizar *big data* para proporcionar estadísticas pasadas, presentes y futuras y conocimiento útil que pueda ayudar a la toma de decisiones y a la mejora del proceso industrial de una compañía. Por tanto, se está contemplando cómo el análisis de los datos puede aportar valor de la forma que se planteaba en el apartado anterior. En la figura 1, se presenta el ciclo de vida de un proyecto basado en la obtención de *big data analytics*.

Figura 1. Ciclo de vida de un proyecto *big data analytics*



Tal y como puede observarse, este ciclo de vida está diferenciado por las siguientes fases:

- **Identificar el problema que hay que resolver**: en todo proyecto *big data*, el paso inicial es plantear el problema o preguntas que se quieren resolver a partir del análisis de los datos. Por poner un ejemplo relacionado con el mantenimiento predictivo, la pregunta sería: *¿puedo predecir cuándo presentará averías la maquinaria?*

- **Identificar los datos necesarios**: una vez identificado el problema que hay que resolver, el siguiente paso es definir los datos que se necesitarán para resolverlo. Cabe mencionar que también es necesario definir la calidad, cantidad, formato y fuentes de los datos. En el ejemplo utilizado, los datos serían *medidas de sensores en tiempo real junto al histórico de medidas anteriores*.

- **Recolección de datos**: el siguiente paso se basa en llevar a cabo la recolección de datos y los mecanismos de obtención de los mismos. En este caso, la recolección puede basarse en la obtención de datos almacenados en bases de datos, flujos de datos provenientes de medidas de sensores en tiempo real o de datos agregados provenientes de pasarelas. En el ejemplo utilizado, se requieren *medidas de sen-*

sores en tiempo real junto al histórico de medidas anteriores almacenado en una base de datos.

- **Preprocesado de los datos**: tal y como se mencionaba en el material «Fundamentos del *big data*: arquitectura del sistema», en un entorno industrial los datos suelen venir desordenados y corruptos, así como de diferentes fuentes. Por otro lado, puede suceder que los datos no se hayan recolectado en el formato necesario. Por este motivo, se requiere una etapa de preprocesado en la que se adapten los datos al formato necesario y se eliminen aquellos que estén corruptos o no presenten la calidad necesaria. En el ejemplo utilizado, *se deberá verificar que todas las medidas presentan valores no corruptos, y deberán combinarse las medidas de los sensores en tiempo real con el histórico, para almacenar todos los datos en un conjunto de datos único (*$dataset$*), utilizando, además, el mismo formato*.

- **Obtención de *analytics***: los *analytics* se llevan a cabo para contestar a las preguntas definidas en el primer paso. Esto requiere una asimilación de las relaciones entre los datos existentes, y esta asimilación se obtiene mediante la aplicación de algoritmos *machine learning*, presentados en el material «Gestión de los datos y su uso: aprendizaje autónomo». En el ejemplo utilizado, los *analytics* obtenidos se referirán al conocimiento generado para contestar a las preguntas: *¿por qué ha ocurrido una anomalía en el pasado?; ¿cómo predigo una nueva anomalía mediante el uso de las medidas de los sensores actuales?*

- **Visualización *analytics***: esta fase se basa en la representación gráfica de los *analytics* obtenidos para mejorar el entendimiento del análisis efectuado o para simplificar la toma de decisiones a partir de los datos. En este caso, se debe tener claro quién será el usuario final, encargado de analizar los datos para llevar a cabo la toma de decisiones, que puede tener un perfil muy poco técnico. Encontramos diferentes herramientas de visualización profesionales, totalmente orientadas a este propósito (como Tableau, Qlikview e incluso Excel). Sin embargo, en muchos casos pueden ser herramientas localmente generadas por los profesionales que han generado los *analytics*. En el ejemplo utilizado, la representación gráfica no resultaría compleja, se basaría en *presentar a los responsables de mantenimiento una señal de alarma cuando se detecta que una maquinaria determinada tiene alta probabilidad de presentar una anomalía*.

Por tanto, la obtención de conocimiento o valor de los datos no proviene de aplicar algoritmos *machine learning* directamente a los datos brutos para observar qué resultados ofrecen, sino que se requiere una fase previa, en la que se necesita experiencia sobre el problema industrial que hay que resolver, para poder plantear el problema adecuado, el tipo de datos necesarios y los algoritmos para obtener la solución. Por este motivo, resolver problemas de este tipo cae más en un diseño $artesanal$ que en uno $mecánico$ y, por tanto, la experiencia del equipo de profesionales involucrado es un gran activo. Para solventar los proyectos con éxito, se requiere la colaboración de equipos multidisciplinares (perfiles más empresariales/industriales relacionados con el problema industrial, haciendo equipo con ingenieros o $data\ scientists$) o contar con

profesionales con perfiles híbridos, basados en tres características: tecnología, ciencia (estadística) y negocio.

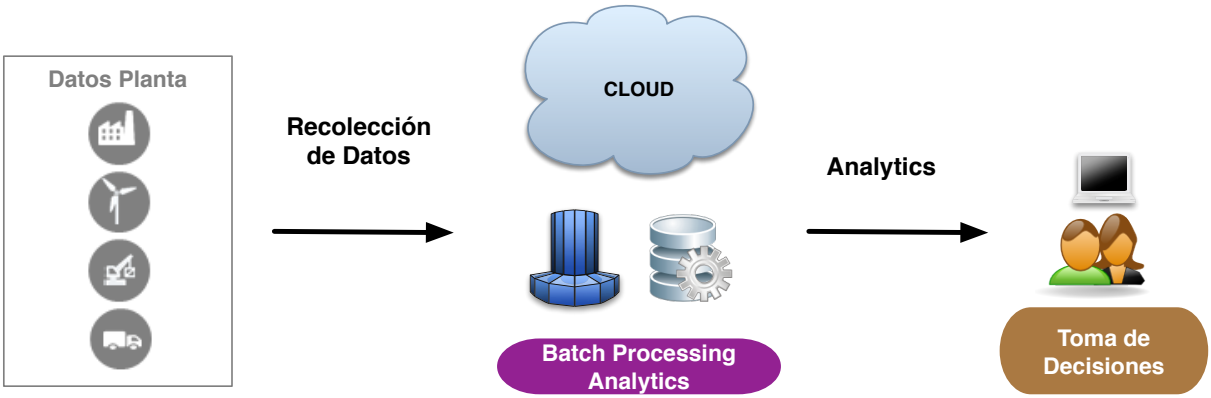
Tal y como se presentó en «Fundamentos del *big data*: arquitectura del sistema», el análisis de datos puede llevarse a cabo o bien en tiempo real, para dar respuesta a actuaciones necesarias con menor lapso temporal, o bien con un periodo temporal mayor, analizando por ejemplo el histórico de datos almacenados en una base de datos local o en la nube. Por este motivo, en este documento se clasifican las metodologías utilizadas en función de si se trabaja con datos en tiempo real o datos posprocesados (*batch processing*), o si se opta por un esquema híbrido conocido como arquitectura lambda. En los apartados siguientes, se presentan estas metodologías.

3. *Batch processing analytics*

3.1. Descripción de la metodología

En este caso, los *analytics* se obtienen mediante el análisis de un volumen alto de datos, almacenado comúnmente en un *cluster* de servidores o en la nube (mediante proveedor externo, como Amazon). Llevar a cabo un análisis con un volumen alto de datos permite obtener unos análisis más elaborados y patrones más precisos. Aparte de esto, se puede disponer de históricos muchos más amplios, tanto en un ámbito temporal como geográfico, y por este motivo, este tipo de análisis suele denominarse como *global analytics*. En el ejemplo de mantenimiento predictivo, se podría llevar a cabo un análisis de datos recolectados de toda la maquinaria en la planta, o en otras instalaciones si la empresa dispusiera de las mismas, y con un histórico que abarcaría un periodo temporal de relevancia (véase la figura 2). Esto permitiría el desarrollo de un sistema predictor con mejores prestaciones, al haberse podido entrenar con una casuística muy completa.

Figura 2. *Batch processing analytics*



Entre las herramientas para llevar a cabo estos análisis, las más utilizadas actualmente son Apache Hadoop (hadoop.apache.org) y Apache Spark (spark.apache.org). Aunque en el material «Tecnología de *big data*» se presentan estas tecnologías con más detalle, el objetivo de este es centrarse en la utilidad de las mismas y las ventajas que pueden ofrecer para la obtención de *analytics*.

Tabla 1

Solución	Ventajas
Hadoop	• Escalabilidad ilimitada gracias a la capacidad de almacenar datos (HDFS), y eficiencia gestionando *clusters* de servidores (YARN). • Apoyo empresarial gracias a la seguridad, encriptación de datos e integridad proporcionada. • Amplio abanico de aplicaciones gracias al apoyo de ficheros (estructurados, semiestructurados y desestructurados), fuentes de *streaming* (Flume y Kafka) y bases de datos (cualquier tipo de base de datos relacional y bases de datos NoSQL).
Spark	• Desarrollo sencillo, gracias a disponer de API nativas en múltiples lenguajes (Java, Scala, Python y R) y apoyo de API de alto nivel (DataFrames, Data sets y Data Sources). • Prestaciones optimizadas. • Unificación (SQL, *machine learning* y *real time processing*).

Cabe mencionar que aunque Hadoop ha sido en los últimos años la solución más popular, el interés por parte de la comunidad *big data* en considerar Spark ha crecido enormemente en los últimos 3 años (superando incluso el interés observado por Hadoop), debido a las ventajas presentadas en la tabla anterior. Por este motivo, en los apartados siguientes nos centraremos en Spark.

3.2. *Machine learning* con Spark

Una de las grandes ventajas aportadas por Spark es el conjunto de algoritmos de *machine learning* que provee mediante la librería MLlib. Concretamente, Spark soporta la ejecución de los algoritmos *machine learning* de forma distribuida, utilizando datos almacenados en diferentes servidores. A continuación, se presentan los algoritmos soportados (podéis ver el apartado de aprendizaje autónomo de este material para obtener una presentación de la teoría de *machine learning* con más detalle).

Tabla 2

Tipo de algoritmo	Tipo de *machine learning*	Nombre de algoritmo
Clasificación	Aprendizaje supervisado	• *Naive bayes* • *Decision trees* • *Random forests* • *Gradient-boosted trees*
Regresión	Aprendizaje supervisado	• *Linear regression* • *Logistic regression* • *Support vector machines*
Clustering	Aprendizaje no supervisado	• *K-means* • *Gaussian mixture* • *Power iteration clustering* (PIC) • *Latent dirichlet allocation* (LDA) • *Streaming K-means*
Dimensionality reduction	Aprendizaje no supervisado	• *Singular value decomposition* (SVD) • *Principal component analysis* (PCA)

Tabla 2. (continuación)

Tipo de algoritmo	Tipo de *machine learning*	Nombre de algoritmo
Collaborative filtering	Sistemas de recomendación	• *User-based collaborative filtering* • *Item-based collaborative filtering* • *Alternating least squares* (ALS)
Extracción y transformación de características	Extracción de características	• TF-IDF • Word2Vec • *Standard scaler* • *Normalizer* • *Chi-square selector*
Optimización	Optimización	• *Stochastic gradient descent* • *Limited-memory BFGS*

3.3. Ejemplo de aplicación

En el ejemplo de mantenimiento predictivo, el objetivo del sistema es predecir si una maquina concreta de la planta industrial puede presentar un fallo en el futuro. Como se ha mencionado anteriormente, la predicción nos permite llevar a cabo un mantenimiento preventivo, lo cual tiene un menor impacto en el funcionamiento de la planta si se compara con un fallo no previsto (lo que puede aumentar el coste de reparación o afectar de forma muy negativa a la actividad de la planta). En la figura 3, se presentan los datos recolectados de diferente maquinaria durante un periodo temporal significativo. Es decir, los valores de M y L serían realmente grandes para tener la mayor casuística posible, y utilizar el mayor número de medidas de sensores disponibles. De ahí que se aborde este ejemplo mediante el uso de *batch processing*.

Figura 3. Datos de sensores de diferente maquinaria

Máquina ID	Sensor 1	Sensor 2	...	Sensor L	Fallo
1	2,633	0,918	...	4,229	no
2	9,244	22,732	...	15,307	sí
3	3,183	28,526	...	8,735	no
4	2,785	2,642	...	5,857	sí
....	...	...	...	...	...
Máquina M	4,889	14,415	...	23,903	sí

Estos datos almacenados serían los utilizados para entrenar el algoritmo de predicción utilizado por el sistema para detectar si una máquina, según las medidas obtenidas en un instante determinado, va a presentar un fallo. Si se opta por la solución Apache Spark, revisando los algoritmos soportados por la misma, algunas posibles opciones que podrían considerarse serían los algoritmos de regresión (utilizados para llevar a cabo tareas de predicción) *logistic refression* o *support vector machines*. El funcionamiento, por tanto, sería el siguiente:

1) **Creación del histórico (datos entrenamiento)**: se van tomando medidas de todos los sensores de todas las máquinas disponibles durante un periodo temporal grande, y se almacenan en la base de datos de la organización o en la nube (base de datos

distribuida si el tamaño de los datos es grande, *big data*), indicando además si se ha producido avería o no en cada uno de los eventos registrados.

2) **Entrenamiento algoritmo de regresión**: se entrena algoritmo *logistic regression* o *support vector machine*.

3) **Monitorización de maquinaria y predicción de anomalías**: continuamente, se van tomando medidas de la maquinaria y se van almacenando en la base de datos. Estas medidas son analizadas por el algoritmo de regresión para evaluar si se va a presentar una avería.

4) **Mantenimiento preventivo**: en el caso de detectar una anomalía, se planifica una acción de mantenimiento predictivo y se reorganiza la planificación de actividades llevadas a cabo en la planta durante este mantenimiento, con el objetivo de reducir su impacto.

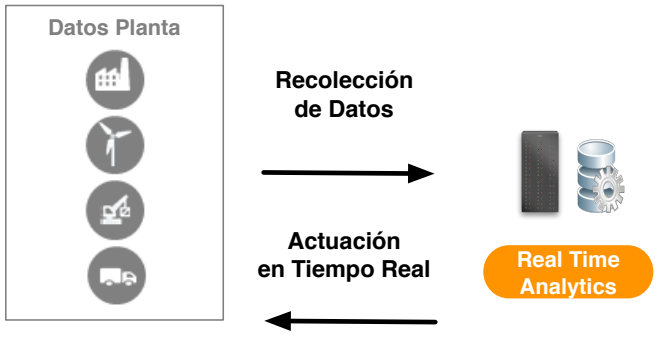
5) **Perfeccionamiento sistema de predicción**: los pasos 1 y 2 se irán ejecutando de forma periódica, para mejorar la capacidad de predicción del sistema a medida que se toman en cuenta nuevas situaciones.

4. *Real time analytics*

4.1. Descripción de la metodología

De nuevo en el material «Tecnologías del *big data*» de este PLA, se mencionan las herramientas utilizadas para procesar datos en modo *streaming* (en tiempo real), pero en este documento se pone énfasis en los aspectos relacionados con la obtención de *analytics*. Cabe mencionar que, inicialmente, las tecnologías *big data*, como Hadoop y Spark, estaban enfocadas a conseguir grandes velocidades de computación, pero no a obtener respuestas con bajos retardos. Por este motivo, han aparecido nuevas herramientas totalmente orientadas a procesar datos de forma continua, a medida que se van generando, con la meta de obtener *analytics* con muy baja latencia (es decir, en tiempo real). Estas herramientas son también muy adecuadas para llevar a cabo actuaciones en tiempo real (véase figura 4).

Figura 4. *Real time analytics* y actuación en tiempo real



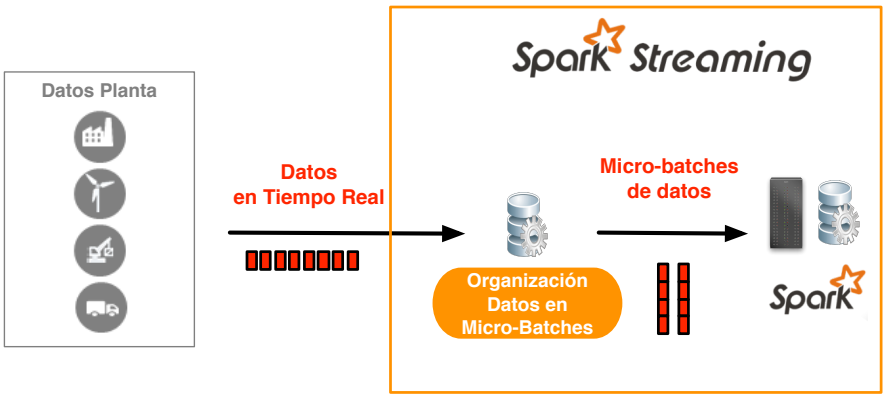
En cuanto a herramientas utilizadas, las más populares en la actualidad son Apache Spark Streaming (spark.apache.org/streaming) y Apache Storm (storm.apache.org): las dos trabajan el concepto de paralelización, y utilizan *clusters* de servidores. A continuación, se muestra una tabla con las ventajas de las dos soluciones:

Tabla 4

Solución	Ventajas
Spark Streaming	• Modelo de programación sencillo (menor coste de desarrollo). • Reusabilidad de código y apoyo de librerías *machine learning*, ya que se basa en Apache Spark (es decir, Spark integra tanto *batch* como *real time processing*).
Storm	• Procesado en tiempo real. • Sistema tolerante a fallos, garantizando que cada dato será procesado. • Soporte más amplio de lenguajes de programación (Java, Scala, Python, Ruby, Clojure).

De la tabla anterior, se puede observar cómo únicamente en el caso de Storm se habla de procesado en tiempo real como ventaja. Esto es así porque Spark Streaming trabaja en un ámbito de *microbatch*, lo que significa que opera *casi* a tiempo real (véase figura 5). En cambio, Storm sí es una solución que opera en un ámbito de datos recibidos en tiempo real. Comparando los niveles de latencias que pueden conseguirse con las dos soluciones se observan diferencias significativas, ya que Storm presenta unas respuestas del orden de 100 milisegundos frente a los 0,5 segundos obtenidos con Spark Streaming. Por tanto, el uso de una herramienta u otra vendrá dado por el compromiso entre latencia (Storm) frente a simplicidad y reusabilidad (Spark Streaming). Cabe mencionar que las dos soluciones se basan en el uso de computación distribuida en *clusters* de computación. Es decir, la capacidad de obtener *analytics* con baja latencia se basa principalmente en la paralelización y, para ello, se requiere una gran experiencia, por parte del desarrollador, para poder configurar esta paralelización de forma adecuada y explotar los recursos disponibles.

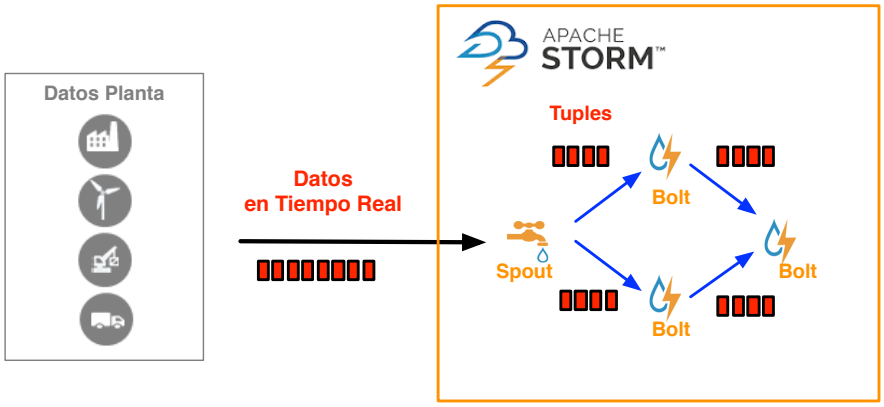
Figura 5. Tratamiento de los datos en *streaming* de Spark



4.2. Arquitectura de Apache Storm

Teniendo en cuenta que Apache Storm es una solución puramente diseñada para llevar a cabo procesado de la información en tiempo real, este subapartado se dedica a presentar la arquitectura utilizada para dicho propósito. En la figura 6, se presenta lo que se conoce como ***topología***, la cual se basa en un grafo de computación en el que cada uno de los nodos representa un mecanismo de computación individual y los arcos representan los datos que viajan entre nodos. Aquí cabe mencionar dos conceptos: 1) el uso de este tipo de metodología se basa en el hecho de que cada nodo lleva a cabo una serie de cálculos (no muy complejos) para procesar los datos con muy baja latencia y pasar el resultado al nodo siguiente; y 2) los datos que van viajando entre nodos se denominan ***tuples***, y cada *tuple* es un conjunto ordenado de valores de tamaño reducido y que se van enviando de forma continua (lo que se conoce como ***stream*** de *tuples*). Estos dos puntos, es decir, procesar información en bloques pequeños, de forma continua, y con mecanismos de computación sencillos desplegados en serie, son los que aseguran que puedan obtenerse resultados con baja latencia.

Figura 6. Topología de Storm



En la figura anterior, uno puede observar también que existen dos tipos de nodos:

- ***Spout***: es la fuente de un *stream* en una topología Storm. Los *spouts* normalmente leen datos de fuentes externas y emiten *tuples* dentro de la topología. Las fuentes externas pueden ser colas de mensajes que se van recibiendo (de Apache Kafka)*, lecturas de una base de datos u otro tipo de fuentes en tiempo real (como por ejemplo, mensajes provenientes de sensores).

- ***Bolt***: a diferencia de los *spouts*, cuyo único propósito es aceptar *tuples* de una fuente, los *bolts* son unos nodos que, aparte de aceptar *tuples*, permiten llevar a cabo tareas de computación o transformación de los datos recibidos. La filosofía de Storm radica en el hecho de que estos *bolts* deben ejecutar operaciones simples y, si se necesitan manipulaciones más complejas, estas se llevan a cabo poniendo diferentes *bolts* en cascada (en la que cada uno de ellos se encargaría de una tarea sencilla).

*** Véase el material «Tecnologías de *big data*» para más detalles.**

Finalmente, cabe comentar que la gran potencia de Storm también radica en la capacidad de paralelización ofrecida por el mismo. Uno puede definir el número de réplicas que quiere para un *bolt* determinado (*bolts* en paralelo que ejecutarán la misma operación) y el motor de Storm ya se encarga, de forma transparente para el desarrollador, de gestionar la paralelización en un ámbito de *cluster* de servidores, para que los *tuples* puedan ir recorriendo la topología en paralelo.

4.3. Algoritmos utilizados

Como se ha mencionado anteriormente, una de las ventajas de Spark Streaming es que puede utilizar los mismos algoritmos que Spark, debido a su arquitectura basada en organizar los datos en *microbatches* en primer lugar, para que estos *microbatches* sean procesados por el motor de Spark. Por tanto, la tabla de algoritmos *machine learning* soportados por MLlib, presentados en el apartado anterior, seguirían siendo válidos

en este caso. El inconveniente, no obstante, es la pérdida de capacidad de procesado a latencias bajas.

En el caso de Storm, tal y como se ha observado, la arquitectura está totalmente enfocada a llevar a cabo procesado en tiempo real. Como se ha comentado anteriormente, se utiliza una topología basada en una distribución de nodos *bolts* en cascada, donde cada uno de ellos se encargan de llevar a cabo manipulaciones no muy complejas, para poder emitir los datos procesados con el menor lapso temporal posible. Algunos ejemplos de manipulaciones comúnmente usadas (en un ámbito de *bolt*) son, por ejemplo, filtrado, agregación de datos o unión de los mismos.

Por tanto, se observa que el propósito de aplicaciones basadas en Storm es obtener algún tipo de información de los datos recibidos o detectar eventos de forma totalmente inmediata. No obstante, el uso de técnicas *machine learning* para sacar mayor conocimiento de los datos o llevar a cabo tareas más sofisticadas también está soportado, pero requieren un tratamiento especial. Por un lado, lo que se contempla es el uso de lo que se conoce como *online machine learning*, en el que, a diferencia del mecanismo *machine learning* tradicional, los datos no se encuentran disponibles por completo (para entrenar el sistema, por ejemplo), sino que van apareciendo de forma secuencial y se van utilizando de manera incremental para adaptar el algoritmo utilizado. Si leemos la frase interior, se identifica que este tipo de algoritmos están alineados con la filosofía *real time*. En este caso, es el desarrollador quien debe implementar las tareas que tiene que ejecutar cada *bolt* para llevar a cabo el algoritmo deseado.

Finalmente, cabe mencionar que existe el proyecto Apache Samoa, en el que se está desarrollando una librería de algoritmos *machine learning* para que se puedan ejecutar en Storm. El proyecto está actualmente en fase de desarrollo, y los algoritmos soportados son: *sequential evaluation task, vertical hoeffding tree classifier, adaptive model rules regressor, bagging and boosting, distributed stream clustering* y *distributed stream frequent itemset mining*.

4.4. Ejemplo de aplicación

Un ejemplo de obtención de *analytics* en tiempo real, para mejorar el funcionamiento de una planta industrial, sería aquel basado en la gestión eficiente de energía. En este sistema, existe una parte que se basa en medir y monitorizar el consumo de energía en diferentes maquinarias y tomas de la planta para detectar desbalance en el consumo, algunos picos o anomalías, etc. Llevarlo a cabo en tiempo real, para poder aumentar la eficiencia energética de manera significativa, en una planta donde existe un alto número de elementos, requiere el uso de una solución que pueda mostrar resultados con baja latencia. Concretamente, en el entorno industrial, ofrecer *real-time* es un requisito fundamental, debido a que cualquier perturbación energética puede afectar a diferentes dominios industriales de forma simultánea, como el consumo global, el proceso de producción o la calidad de los productos. Utilizando la solución Storm, el sistema iría calculando los *streams* de datos de consumo energético de toda la planta. La topología

estaría formada por un conjunto de *spouts* recolectando datos de consumo y una serie de bolts llevando a cabo las siguientes tareas (una tarea por *bolt* de forma secuencial):

1) **Filtrado de datos**: para atenuar posibles efectos de ruido en las medidas.

2) **Normalización**: para poder comparar medidas de diferentes dispositivos de forma justa, según el rango de consumo de cada uno.

3) **Muestreo**: para quedarnos con un conjunto de muestras según el periodo de la aplicación.

4) **Selección de los elementos que más consumen en la planta**: se seleccionaría el top P de consumidores para poder llevar a cabo acciones de corrección, siendo P un parámetro de la aplicación concreta.

Los ajustes de paralelización de los *bolts* ya vendrían dados según los requisitos de la aplicación, el número de fuentes de medidas de consumo energético y la capacidad del *cluster* de servidores utilizado.

5. Arquitectura lambda

Encontramos aplicaciones en las que se requiere dar respuesta en tiempo real pero, a su vez, poder explotar un conjunto de datos mayor para proporcionar unos *analytics* con mayor valor añadido.

Por poner un ejemplo, se puede considerar el caso en el que se controla la temperatura de una planta mediante el uso de un conjunto de sensores instalados de manera distribuida en la misma. A continuación, se muestra qué tipo de procedimiento se llevaría a cabo según la estrategia de procesado de los datos:

- Optando por una solución puramente operando en tiempo real, se irían tomando lecturas de los siguientes sensores y aumentando o disminuyendo la temperatura según la tendencia observada en las últimas medidas. Este procesado de información es el que se llevaría a cabo en lo que se conoce como ***speed layer*** (capa rápida). El inconveniente es que la respuesta ofrecida por el sistema quizá no es la más apropiada.

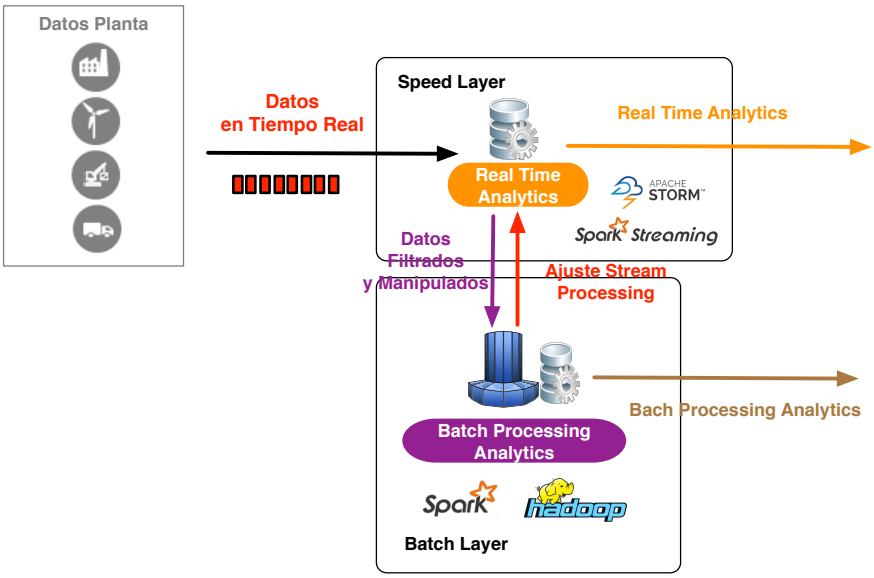
- Si se utilizara un algoritmo puramente basado en *batch processing*, se observaría cómo las decisiones tomadas partirían de un histórico mayor de datos y serían más eficientes. Es decir, si se tuviera en cuenta el histórico de las temperaturas de la planta, se podría detectar si realmente la temperatura que se está percibiendo de los sensores es correcta, se debe a un evento especial (gran aglomeración de personas) o a un fallo del sensor. A partir de esto se adaptaría la actuación del sistema, lo que se conoce como ***batch layer***. El inconveniente es que la respuesta ofrecida por el sistema no sería en tiempo real.

- Una solución híbrida sería aquella que va tomando decisiones con las últimas medidas en tiempo real, pero contrastando los resultados obtenidos con aquellos provistos por un sistema de computación *batch processing*. Es decir, el objetivo es que el mecanismo utilizado en la capa *speed layer* se vaya adaptando y corrigiendo para mejorar las prestaciones del sistema.

La solución híbrida propuesta en el tercer punto es lo que se conoce como ***arquitectura lambda***. El diseño de esta arquitectura se basa en la capacidad que tiene para llevar a cabo un procesado de datos a tiempo real, así como la eficacia obtenida mediante el procesado de datos (en periodos de mayor duración, pero en paralelo) existentes en una base de datos mayor. Para ello, la arquitectura utiliza la estructura definida en la figura 7, en la que se puede observar cómo está formada por dos capas, la capa rápida (*speed layer*), que se encarga de llevar a cabo un procesado en tiempo real, y la capa *batch* (*batch layer*), que lleva a cabo el *batch processing*. Podéis observar en la figura

cómo la solución lambda se basa en la coexistencia de soluciones en *real time* y *batch processing* presentadas en apartados anteriores.

Figura 7. Arquitectura lambda



Resumen

En este material, se ha puesto énfasis en la quinta V de *big data*, el valor de los datos. Por este motivo, se ha presentado el concepto de *big data analytics*, que hace referencia al concepto de obtener estadística y conocimiento de los datos para mejorar los procesos industriales. Además, se han planteado los diferentes pasos de un proyecto de *analytics* y se ha hecho hincapié en la necesidad de disponer de experiencia de carácter multidisciplinar (tecnología, ciencia y negocio) para poder plantear las metodologías de análisis adecuadas.

Una vez presentado el concepto, se ha pasado a describir los diferentes mecanismos utilizados para obtener *analytics*: *batch processing* (posprocesado) y *real time* (*streaming*). Aparte de plantear diferentes herramientas y ejemplos de aplicación, se ha mostrado el compromiso entre prestaciones obtenidas frente a retardo de la respuesta. Con el fin de abordar este compromiso, existe una metodología híbrida, conocida como arquitectura lambda, la cual se ha presentado también para cerrar este material.

Bibliografia

Allen, S.; Jankowski, M.; Pathirana, P. (2015). *Storm Applied*. Manning Publications.

Bibliografía

Allen, S.; Jankowski, M.; Pathirana, P. (2015). *Storm Applied*. Manning Publications.

Ankam, V. (2017). *Big Data Analytics*. Packt Publishing Ltd.