UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Máster en Big Data y Data Science: ciencia e ingeniería de datos

TRABAJO FIN DE MÁSTER

Implementación de una arquitectura Lambda con Spark orientado a IoT

Pablo Machío Rueda
Tutor: Paulo Villegas

Marzo 2023

Implementación de una arquitectura Lambda con Spark orientado a IoT

AUTOR: Pablo Machío
TUTOR: Paulo Villegas

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Marzo de 2023

Resumen

Este Trabajo Fin de Máster tiene como finalidad mostrar cómo se podría implementar una arquitectura Lambda orientada al análisis de los datos de un servicio IoT que adquiere datos de telemetría de vehículos, siendo Spark la herramienta que vertebra el análisis en las distintas capas de la arquitectura

Primero se introducirá brevemente IoT y la arquitectura más común empleada, seguido de la introducción del concepto de BigData y cómo encaja en la capa de análisis de la arquitectura IoT, y se razonará por qué se selecciona la arquitectura Lambda Unificada de BigData y cómo Spark facilita su implementación y se exponen unos casos de uso relacionados con la telemetría de vehículos para ilustrar está implementación.

A continuación, se presentarán tres casos de uso para ilustrar la arquitectura Lambda aplicada al análisis de vehículos de combustión gracias a la información obtenida de la conexión al diagnóstico de a bordo del vehículo.

Después se muestra el diseño de la arquitectura empleada y se comentan las herramientas utilizadas, así como su implementación final.

Por último, se realiza una evaluación de la arquitectura desarrollada teniendo en cuenta las facilidades ofrecidas

Agradecimientos

Agradecer a mis padres y amigos por el apoyo y la paciencia que han tenido durante todo este tiempo.

También me gustaría agradecer a los profesores del master por el conocimiento compartido, especialmente a Paulo, mi tutor del proyecto, por haber respondido a mis dudas con tanta celeridad.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

El Internet de las cosas o IoT por sus siglas en Ingles [1], permite crear una representación virtual de objetos físicos mediante:

- Sensores que miden el contexto de estos objetos físicos
- Actuadores que permiten la interacción con los mismos.

IoT vertebra la comunicación entre los objetos y hacia ellos, y para poder explotar su potencial, se base usa una arquitectura dividida en capas [1] [2], donde las primeras capas se ocupan de los dispositivos que miden el mundo físico, la comunicación entre ellos y el exterior, otras capas son dedicadas a recolección de los datos y su análisis, y finalmente unas capas de aplicación y negocio para obtener conocimiento de acuerdo al dominio deseado.

El gran volumen de datos que generan los sensores hace que el Big Data la tecnología seleccionada para la almacenar de datos y realizar el análisis de éstos [2]. Big Data está pensado para trabajar con volúmenes de datos grandes y complejos [3] que no podríamos gestionar con sistemas incrementales clásicos [4]. El Big Data se fundamenta en el uso de distintas tecnologías combinadas que se armonizan bajo una arquitectura. A lo largo de la vida del Big Data han surgido distintas arquitecturas [3], aunque las más extendidas son Kappa y Lambda. La elección de la arquitectura a emplear depende del caso de uso.

1.1 Motivación

Esta memoria de TFM está motivada por la dificultad de elección de una arquitectura BigData ya que la decisión depende del uso que se le quiera dar. En este caso se va a aplicar una arquitectura Lambda orientada al análisis de datos de dispositivos conectados al puerto del sistema de diagnóstico de vehículos de combustión como coches o furgonetas, que da acceso a los distintos sensores del vehículo.

1.2 Objetivos

El objetivo principal usar parte de los datos de telemetría y posición adquiridos por los dispositivos para optimizar el consumo de combustible de vehículos. Concretamente, se usará:

- Las localizaciones, que se usarán para establecer trayectos realizados por el vehículo con el fin de realizar comparaciones de combustible consumido y distancia recorrida en los trayectos etiquetados como iguales.
- El odómetro que permite conocer la distancia total recorrida por trayecto.
- El combustible consumido que permite conocer el combustible total consumido por trayecto.
- El porcentaje de acelerador usado, que ayuda detectar comportamientos de conducción poco eficiente.
- El nivel del tanque de combustible, que ayuda detectar comportamientos de conducción poco eficiente y robos de combustible.

Para poder realizar estos análisis se diseñará e implementará una arquitectura Lambda para Big Data usando el ecosistema *Spark* y tener así un repositorio único de código para el análisis de datos, reduce la complejidad de la arquitectura y facilita el poder aplicar las

operaciones de análisis, así como el entreno de modelos de Machine Learning, algo que es más difícil de realizar en Kappa por estar orientada al tiempo real [5].

1.3 Organización de la memoria

Para conseguir el objetivo propuesto primero estableceremos las bases sobre las que trabajaremos introduciendo como se hace actualmente el análisis de trayectos, así como la estimación del consumo de combustible en los vehículos de combustión. También se introducirá el concepto de IoT y su arquitectura más extendida. La introducción se completará con la idea de que es el BigData y las arquitecturas más extendidas. Y finalmente se expondrá el procesamiento en Batch y Streaming que forman parte de las arquitecturas Kappa y Lambda. Por último, en este punto, se introducen las tecnologías que se van a usar en la implementación.

Una vez expuesta base teórica se presentará la prueba de concepto (PoC) que servirá como ejemplo para ilustrar del diseño e implementación de la arquitectura Kappa. El PoC está orientado a la explotación de datos de telemetría de vehículos recolectados gracias a una infraestructura IoT y se divide en tres casos de uso.

Luego se establecerá el diseño de la arquitectura, sus componentes y las funciones de los componentes en la arquitectura, así como las tecnologías usadas en cada componente.

Después, se detalla la implementación de cada caso de uso. Por cada caso de uso se indica la capa implicada de la arquitectura, así como el origen y destino de los datos, la estructura de datos de datos de entrada y salida de cada transformación, y la lógica de la transformación y sus test unitarios que ayudan a su compresión. Se hace referencia al repositorio que contiene un archivo docker-compose para levantar el entorno y una serie de scripts para ejecutar la prueba, todo ello detallado en el README.md del repositorio.

El último apartado contiene las conclusiones y se comentan posibles trabajos futuros.

2 Estado del arte

2.1 Internet de las cosas (IoT)

El internet de las cosas o IoT, es un tipo de red cuya finalidad es interconectar, mediante protocolos estandarizados, dispositivos capaces de obtener información (sensores), como pueden ser un localizador GPS, y dispositivos capaces de realizar una acción (actuadores), como puede ser una salida digital que activa el corte de motor de un vehículo, mediante Internet. Esta interconexión de dispositivos es la base para tareas más complejas como puede ser análisis de datos para distintos modelos de negocio, toma de decisiones, ejecución de acciones en función del contexto, reconocimiento inteligente, posicionamiento, monitorización... [1]

Las tecnologías que permiten IoT se pueden categorizar en:

- Tecnologías que adquieren información contextual.
- Tecnologías que permiten el procesamiento de la información contextual adquirida
- Tecnologías para mejorar la seguridad y la privacidad.

Las 2 primeras son los bloques funcionales que permiten dar inteligencia a las cosas.

El tercer bloque, aunque no es funcional, es una necesidad de facto para permitir el uso de IoT [1].

Podemos ver que el IoT no es una sola tecnología, sino una mezcla de distintas tecnologías software y hardware. Es una gran mezcla de tecnologías que necesitan ser adaptadas para permitir aplicaciones de IoT como la eficiencia energética, velocidad seguridad, confianza [1].

Estas tecnologías se agrupan por capas dando lugar a la arquitectura de IoT, donde el número de capas puede variar según como se realice la agrupación, pero es muy común la agrupación en 4 capas: capa de dispositivos y sensores, capa de gateways y redes, capa de gestión de servicios y capa de aplicación [2]. Las capas se comunican con sus capas adyacentes. De la más baja a la más alta tenemos:

- **Capa de dispositivos y sensores**. Conexión entre el mundo físico y el digital, capturan la información mediante sensores y puede realizar un pequeño procesamiento sobre ésta. También se encarga aplicar las decisiones sobre los elementos físicos mediante actuadores.
- **Capa de gateways y redes.** Se encarga del transporte de la información capturada y tiene que lidiar con la heterogeneidad de los dispositivos mediante la unificación protocolos.
- **Capa de gestión de servicios.** Se encarga del almacenamiento, análisis de la información capturada, controles de seguridad, modelado de procesos y gestión de dispositivos.
- **Capa de aplicación.** Aplica toda la información al dominio del problema en cuestión.

El prototipo se encontraría en la capa de gestión de servicios, realizando el análisis de los datos y generando nueva información a partir de estos

2.2 Análisis de trayectos Big Data

Se entiende por trayecto al recorrido realizado entre un punto de origen y uno de destino, en nuestro caso los puntos de origen y destino están ligados al estado del motor del vehículo,

siendo el origen el punto en que el motor se enciende y el destino el punto en el que el motor se apaga, todo lo que pase fuera de esos dos estados no interesa para nuestro. Existen muchas maneras de filtrar dentro de grandes volúmenes los datos relacionados con trayectos con el fin de buscar patrones y comportamientos recurrentes [8]:

- Por origen y destino.
- Análisis espacial
- Agregación de puntos sobre pequeñas celdas geográficas para un gran número de celdas.

Para la selección de modelos predictivos en conjuntos de datos relacionados con trayectos es importante identificar las variables explicativas y los mecanismos físicos [8]. La selección de los modelos se base en aproximaciones entre modelos lineales o modelos basados en la regresión del kernel [8].

Para el PoC se usa regresión lineal para clasificar los trayectos con un origen y destino igual para poder realizar la comparación entre ellos.

2.3 Estimaciones de consumo

Los modelos de estimación de combustible de décadas pasadas como los modelos americanos MOBILE y MOVES de la EPA, o el modelo europeo COPERT se basaban en las características de los vehículos como su motores o potencia, pero no eran muy exacto y se podían mejorar en base al posicionamiento GPS de los vehículos [6]. Combinando los modelos anteriores y el posicionamiento GPS se mejora la precisión de las predicciones [7]. Actualmente existen en el mercado dispositivos que se conectan directamente al puerto de diagnóstico del vehículo (On Board Diagnostics o OBD, cuyo protocolo actual es OBDII) para obtener los valores de telemetría del vehículo, como el consumo instantáneo o el acumulado, y transmitir estos datos a un móvil mediante Bluetooth o directamente a un servidor mediante redes móviles. En definitiva, dispositivos IoT que capturan la telemetría del vehículo.

Para el prototipo se hará uso de la información obtenida directamente del vehículo gracias a dispositivos conectados al puerto OBDII del vehículo.

2.4 BigData y arquitecturas

La idea del Big Data está fuertemente relacionada el abaratamiento del coste de almacenamiento de datos y la gran cantidad de datos estructurado, semiestructurados y no estructurados que generan las aplicaciones actuales o que las empresas tienen ya almacenados [3].
Trabajar con estos datos y obtener conocimiento de ellos es muy complicado ya que los conjuntos de datos se pueden encontrar en distintas escalas, pueden ser adquiridos de distintas formas, pueden tener distinta naturaleza (estructurados, semiestructurados, y no estructurados) [3]. El big data pueden caracterizarse, por tanto, por la creación de datos y su almacenamiento, análisis y recuperación, tal y como se define en 5 V

- **Volumen.** Denota la gran cantidad de datos generado en poco tiempo que determina el valor y el potencial que se deben considerar y requiere de plataformas de computación especial en orden de poder analizarla.

- **Velocidad.** Se refiere a la velocidad en la que los datos son generados y procesados para poder superar los retos y la demanda que reside en el camino del desarrollo y crecimiento
- **Variedad.** Se refiere al tipo de contenido para el análisis de datos. Los datos provienen de distintas fuentes y por tanto es heterogéneo en su contenido.
- **Veracidad.** Para algunas organizaciones la calidad de los datos **variables** es básica.
- **Valor** Se enfoca en lo significativos que son los datos almacenados para su análisis.

El Big Data no se pude alcanzar con herramientas convencionales como las bases de datos relacionales, sino que requiere la combinación de varias herramientas formando una arquitectura para el almacenamiento y análisis de los datos [4].
Existen múltiples arquitecturas y nuevas arquitecturas están surgiendo con el uso de la Nube, pero las más antiguas y extendidas son Lambda y su versión simplificada, Kappa [3] [4].

2.4.1 Arquitectura Lambda

En la arquitectura Lambda se busca poder ejecutar consultas genéricas sobre todos los datos disponibles, tanto históricos como los que se van adquiriendo en tiempo real. Como el volumen de datos es muy grande, una manera de reducir el tiempo de las consultas es calcular los datos previamente y creando vistas sobre las que se realizan las consultas, pero como como son muchos datos y la creación de las vistas lleva tiempo, es muy probable que nuevos datos lleguen al sistema mientras se crean las vistas. Para solventar este problema la arquitectura se compone de 2 flujos de datos distintos, uno corresponde a un flujo de datos potencialmente infinitos sobre el que se aplica un procesamiento en Streaming, y el segundo sobre un conjunto de gran volumen datos estáticos ya asentados al que se le aplica un procesamiento por lotes o batch. La combinación de ambos flujos permite trabajar con volúmenes de datos propios del BigData con latencias bajas. Los flujos son complementarios, ya que el procesamiento batch requiere de mucho tiempo y por tanto tiene una latencia alta que impide que mantener los datos actualizados, para solventar este problema se usa el procesamiento en Streaming sobre los datos que van llegando al sistema mientras aún se está ejecutando el procesamiento en batch. Aunque el procesamiento en Streaming puede ser imprecisos por la naturaleza de un sistema distribuido donde los datos se procesan en tiempo real, sus resultados son una buena aproximación hasta que estos datos son tenidos en cuenta por el procesamiento en batch y sustituidos por los resultados de éste. Para conseguirlo, la arquitectura Lambda se separa en 3 capas. La Capa batch y la Capa de servicio se ocupan del flujo de procesamiento sobre el gran volumen los datos estáticos, y la Capa de tiempo real se ocupa del flujo de procesamiento de los datos en tiempo real [4].

2.4.1.1 Capa Batch

Es responsable tanto de persistir los datos maestros o datos brutos, así como de ejecutar los procesos batch para crear las vistas que se almacenarán en la Capa de servicio para su posterior consulta. La persistencia de los datos maestros es inmutable, una vez guardados no se puede modificar, para evitar la corrupción de estos. Es fundamental que no se puedan cambiar porque así se previenen fallos humanos sobre el procesamiento para crear las vistas. Si el procesamiento no se implementó correctamente, las vistas se pueden borrar y volver a crear con una versión corregida [4].

Para la persistencia se suelen usar sistemas de archivos distribuidos como HDFS
Para el procesamiento batch se solía utilizar Hadoop MapReduce, aunque ahora es más común usar Databricks, Apache Spark o Apache Flink.

2.4.1.2 Capa de Tiempo real.

Esta capa se encarga de procesar los datos que entran en el sistema en tiempo real para estos se puedan consultar mientras la Capa batch está generando las vistas que aún no están disponibles en la Capa de servicio para ser consultadas [4]. Algunas de las tecnologías que se pueden usar en esta capa son: Spark (Structured Streaming), Flink, Storm, Kafka Stream

2.4.1.3 Capa de Servicio.

Se encarga de guardar las vistas generadas en la Capa batch y permitir su consulta de forma eficiente. Se suelen usar bases de datos NoSQL que fomenten la lectura de datos por encima de la escritura ya que los datos no se actualizarán, si es necesario se borrarán y se volverán a crear [4]. Algunos ejemplos son: Cassandra, CouchDB ...
[4].

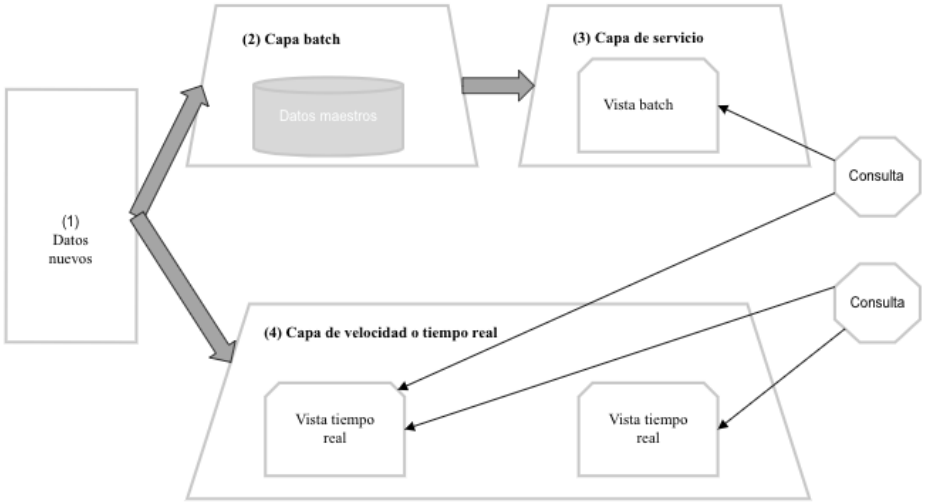

Ilustración 2-1: Arquitectura Lambda

El problema de la arquitectura Lambda es que exige el mantenimiento de dos repositorios de código, uno para la capa Batch y otro para la capa de Tiempo real. La complejidad radica en que las estructuras de datos de ambas capas deben permanecer sincronizadas. Para solucionar este problema surge la arquitectura Kappa que busca simplificar la arquitectura Lambda al quitar la capa Batch y quedarse solo con las capas de Tiempo real y Servicios, y por tanto solo existe un repositorio, simplificando las tareas de desarrollo como depuración o el mantenimiento. Por el contrario, arquitectura Kappa carece de la potencia de cómputo que tiene la capa Batch, y tareas como el entrenamiento de modelos de Machine Learning o la regeneración de las vistas se ven penalizados [5].

La elección de que arquitectura usar depende del tipo de uso que se le vaya a dar [3][5].

En el caso del análisis de vehículos mediante la información obtenida por el puerto OBD, como es el caso del prototipo que se va a implementar, el uso tanto de la capa de tiempo real (Streaming) como de la capa Batch permite obtener el mayor conocimiento de los datos ya que la ejecución de procesos de por lotes al poder trabajar sobre todos los datos históricos [8], por tanto, la arquitectura Lambda sería la que mejor se ajustaría.

2.5 Procesamiento Batch y Streaming

Para entender la diferencia entre ambos tipos de procesamiento datos entender que los datos tienen dos dimensiones principales [10]:

- La cardinalidad, que determina el tamaño del conjunto de datos y que puede ser un tamaño limitado o un tamaño ilimitado donde los datos son potencialmente infinitos.
- La constitución de los datos o como se manifiestan los datos. Estos se pueden manifestar de una forma estructurada como una tabla o visto como una evolución en el tiempo elemento a elemento del conjunto de datos, que se conoce como Stream.

También es importante entender los dominios de tiempo existentes en el procesamiento de datos [10]. Los dos dominios más importantes son:

- El momento del evento. El momento del evento corresponde al instante de tiempo en el que se genera el dato o evento.
- El momento del procesamiento El momento de procesamiento corresponde al instante del tiempo en que el dato o evento es observado por el sistema.

Si el conocimiento que queremos obtener de los datos es en base al momento del evento, nos podemos encontrar el problema de que los datos estén desordenados, y en el caso de trabajar con una cardinalidad en la que los datos tienen un tamaño ilimitado, no podamos determinar si tenemos todos los datos para poder procesarlo de forma ordenada.

En el caso del procesamiento Batch [10], se trabaja con un conjunto de datos de tamaño limitado, ya que no entrarán nuevos datos durante el procesamiento, y los datos se encontrarán estructurados, normalmente en una tabla y no hay ningún problema para trabajar con los datos en base al momento del evento ya que todos los datos con los que se va a trabajar ya están presentes.

En el caso de procesamiento en Streaming [10], se trabaja con un conjunto de datos de tamaño ilimitado, y los datos están en forma de Stream. En este caso si hay un problema para trabajar en base al instante de evento ya que los datos pueden llegar desordenados y no hay garantía de que hayamos visto todos los datos. Cuando en Streaming se trabaja con datos en base al instante del evento es necesario guardar el estado actual de los datos que hemos que ya han sido procesados y también es necesario tener herramientas que nos permitan establecer un criterio por el que determinar si se han visto o no todos los datos. No se puede determinar de forma exacta si se han visto todos los datos ya que estos son potencialmente infinitos, por eso el procesamiento en Streaming es menos preciso que el procesamiento Batch.

2.6 Apache Spark

Apache Spark es un motor unificado de procesamiento y análisis de datos orientado al Big Data que permite trabajar con múltiples lenguajes de programación como Scala, Python o R y que permite realizar procesamiento de datos tanto en Batch como en Streaming. Spark también dispone de librería para entrenar y aplicar modelos de Machine Learning, trabajar con grafos o realizar análisis de datos empleando SQL como si fuera una base de datos.
Es un proyecto Open Source mantenido por la fundación Apache.

2.7 Apache Kafka

Apache Kafka es un sistema distribuido de streaming de eventos, altamente escalable y tolerante a fallos que permite trabajar con flujos de datos en tiempo real.
Es un proyecto Open Source mantenido por la fundación Apache.

2.8 Docker y Docker-Compose

Docker es una herramienta contenerización que permite la ejecución de aplicaciones o servicios en un mismo sistema operativo de forma aislada.
Es sencillo de utilizar gracias al uso de un lenguaje declarativo que permite la definición de la infraestructura como código.
Docker-Compose es una herramienta de Docker que permite desplegar varios servicios simultáneamente y que puede ser aislados dentro de su propia red virtual de forma que no interfieren con otros servicios que se estén utilizando en el sistema operativo anfitrión.

3 Diseño

3.1 Introducción

A continuación, se expone el prototipo en Spark que se va a implementar junto con sus casos de uso.
Finalmente se detalla la arquitectura Lambda con Spark para llevar la implementación del prototipo.

3.2 Prototipo

Para mostrar cómo sería la implementación de la arquitectura Lambda usando Spark vamos a basarnos en aplicación IoT en la que se instalan, al puerto OBD de vehículos de combustión, unos dispositivos que leen los datos disponibles de los propios sensores de telemetría del vehículo y transmiten esos datos en un formato unificado basado en JSON. La unidad de información la llamaremos Frame y está compuesta por los valores actuales de los sensores, la localización del vehículo, la fecha de recolección por parte del dispositivo y la fecha de entrada en el sistema. Más detalle en el apéndice, apartado Modelo de datos, sección Frames. El tamaño medio de los frames es 1,4KB, dependiendo de los sensores disponibles en el vehículo, que se suelen transmitir con una frecuencia de uno por minutos aproximadamente, generando unos 2 MB al día por vehículo.

3.2.1 Casos de uso

El análisis se realiza sobre la manera de conducir del conductor, para ello se agregarán los frames en trayectos realizados por el vehículo. Los trayectos serán categorizados con fin de poder comparar trayectos de una misma categoría entre sí.
Como una parte determinante del consumo es la manera de conducir, se generan una serie de eventos relacionados con la conducción del vehículo de manera que se puedan contabilizar las ocurrencias de los eventos por trayecto. En este caso, el análisis se centrará solo en dos de los posibles eventos, el porcentaje de aceleración y otro en base al consumo brusco de combustible.

3.2.1.1 Creación de trayectos

Un trayecto es una agregación de frames, compuesta por la fecha y localización de inicio, la fecha y localización de fin, la diferencia entre la distancia final e inicial, y la diferencia del consumo final e inicial. Su creación queda determinada por los cambios del sensor de la llave de ignición. Cuando ésta pasa de apagado a encendido se considera que se ha iniciado el trayecto, cuando pasa de encendido a apagado se considera que ha terminado.

El sistema será capaz de detectar los cambios de estado de ignición y agregará los valores de todos los frames que se consideran dentro de un trayecto. Los campos del trayecto son:

- La localización y fecha del inicio del trayecto. Corresponde al primer frame con la llave de ignición encendida.
- La localización y fecha del final del trayecto. Corresponde al último frame con la llave de ignición encendida.

- La distancia total recorrida. Diferencia del odómetro del último frame con la llave de ignición encendida respecto del primer frame con la llave de ignición apagada.
- El consumo total realizado. Diferencia del total de combustible consumido del último frame con la llave de ignición encendida respecto del primer frame con la llave de ignición apagada.

3.2.1.2 Etiquetado de trayectos.

No tiene sentido comparar todos los trayectos entre sí, sino que deben ser categorizados en función de su localización de inicio y localización de fin, por ejemplo, una categoría sería la de los trayectos que se hacen de casa al trabajo y otra sería la de los trayectos que hacen del trabajo a casa. No importa el recorrido que se haga entre medias ya que para mejorar la optimizar el consumo de combustible, tan importante es la manera de conducir como el recorrido realizado.
Para el etiquetado se usará un modelo de Machine Learning supervisado, cuyas etiquetas decide el usuario en base a los trayectos que se han realizado previamente que y que considera susceptibles de mejorar en lo que ha consumo concierne. Los trayectos que no son etiquetados quedan todos en la misma categoría, aunque está no es útil para comparar por la naturaleza heterogénea de los trayectos que la componen.

3.2.1.3 Creación de evento de exceso de aceleración.

Uno de los factores determinantes en el consumo de un vehículo es el uso que se realiza del acelerador [3]. Cuando se comparan trayectos es importante saber si el conductor de vehículo hizo aceleraciones bruscas, ya que suponen un mayor consumo del combustible.
Los frames recogen el valor correspondiente al sensor del acelerador, y teniendo en cuenta que un porcentaje de pisada del pedal superior al umbral del 20% se considera una aceleración brusca, el sistema puede generar un evento por cada frame que contenga un valor superior a este umbral.

3.2.1.4 Creación de evento de descenso brusco del nivel de combustible.

Un descenso brusco del nivel de combustible es un indicador claro de un consumo inapropiado de combustible. A parte de que se pueda relacionar una conducción poco eficiente, si se produce fuera de un trayecto puede indicar un robo de combustible.
Los frames recogen el valor correspondiente al sensor del nivel de combustible. Un descenso superior al 5% en un rango de tiempo inferior a 5 minutos indican un descenso brusco. Como el evento no es solo importante para el análisis de la conducción sino también para la monitorización de robos de combustible, el evento se debe generar en tiempo real.

3.3 Arquitectura aplicada

La arquitectura lambda que se va a implementar para el prototipo está basada en Apache Spark, que es un motor multilenguaje orientada al procesamiento distribuido de datos. Spark proporciona herramientas tanto para la Capa batch como para la Capa de tiempo real, de manera que podemos usar un mismo lenguaje para ambas capas, fomentando la mantenibilidad y la consistencia en la aplicación de lógicas de transformación y análisis de datos originados por distintas fuentes.

Para simplificar el trabajo, se da por supuesto que la Capa batch ya ha se ha encargado de persistir los datos crudos siguiendo una estructura determinada de directorios y el prototipo se limitará al consumo de está mediante la lectura un archivo json plano para emular la fuente de información. Los resultados de la capa se guardan directamente en PostgreSQL

En el caso de la capa de tiempo real, se usará Apache Kafka como fuente de datos. Apache Kafka es un sistema de mensajería distribuida que se suele usar como fuente de Streaming. En el caso del prototipo, contiene los frames que procesan en tiempo real. Spark permite trabajar con fuentes de streaming, como Kafka, mediante Spark Structured Streaming. Spark Structured Streaming abstrae parte de la complejidad de trabajar con streams permitiendo crear pipelines de una manera muy similar a la que ofrece para procesamiento en batch. Como sink del pipeline se utiliza PostgreSQL

En la Capa batch tiene lugar los casos de uso de creación de trayectos, el etiquetado de trayectos y los eventos de conducción excesiva ya que son funcionalidades que no requieren una inmediata notificación a los usuarios y están destinados a un análisis pausado de la información.

En la Capa de tiempo real tiene lugar el caso de uso del evento de descenso brusco de combustible ya que no solamente puede indicar una conducción sino también un robo de combustible, haciendo necesaria una intervención inmediata.

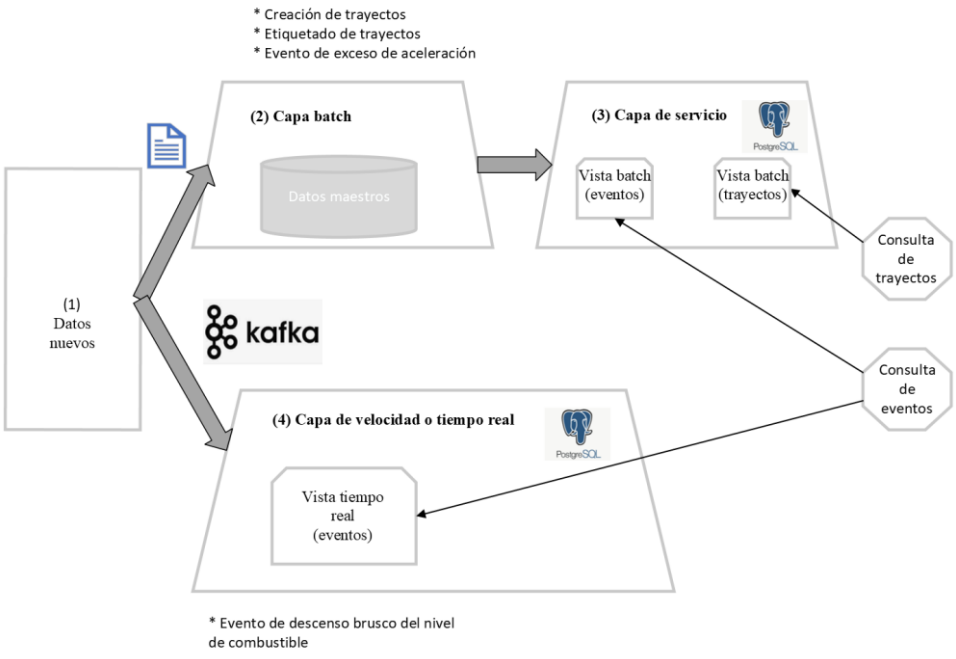

Ilustración 3-1: Implementación arquitectura lambda

Las consultas se realizan mediante una aplicación web muy sencilla que ofrece una API basada OpenAPI mediante Swagger para facilitar que terceros puedan usar el API para consultar las vistas. En el caso de prototipo, tanto las vistas de la Capa tiempo real como de la Capa batch se guardan como tablas dentro del PostgreSQL.

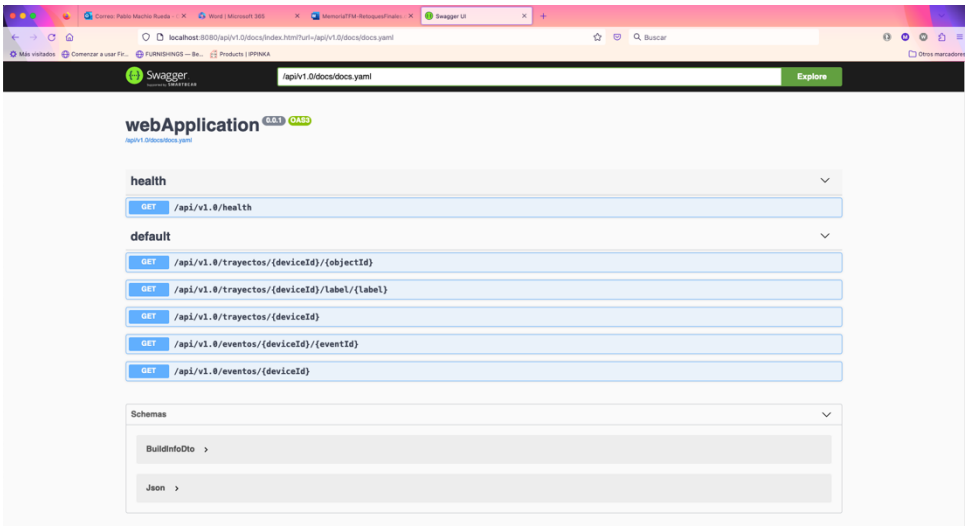

Ilustración 3-2 Vista general de la aplicación web

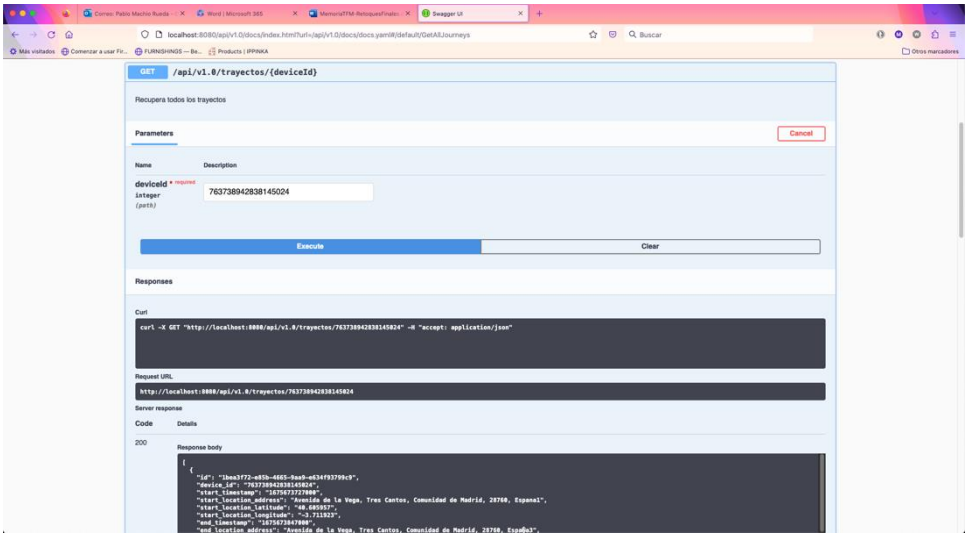

Ilustración 3-3 Vista de la búsqueda de trayectos por dispositivo

4 Desarrollo

Todo el desarrollo se encuentra recogido en el repositorio de GitHub https://github.com/pmachiouam/MasterBigData_PF al que se hará referencia para ilustrar el desarrollo sin tener que añadir todo el código. Está estructurado de la siguiente manera:

- "raiz": Contiene los scripts para arrancar el entorno de ejecución del prototipo. Los scripts están enumerados en base al orden en que deben ejecutarse. Existe un archivo README.md que detalla los scripts y la forma de ejecutarlo. Dentro de la carpeta raíz encontramos las siguientes subcarpetas:
    - "codigo". Contiene todo el código en Scala estructurado según la herramienta de construcción SBT.
        - "project". Contiene los archivos con las referencias a las dependencias del código que usa SBT para compilar y construir los artefactos de los distintos proyectos, así como los plugins que necesita SBT para construir los artefactos.
        - "spark_proj". Contiene todo el código del prototipo de la arquitectura. El código de cada capa se detalla en los siguientes puntos.
        - "web". Contiene la web que se encarga de consultar a las vistas generadas por ambas capas.
    - "documentacion". Contiene documentos relacionados con la realización de la memoria.
        - "Trabajos de Otros". Contiene la mayoría de los trabajos a los que se hace referencia y otros proyectos que han servido de base para el desarrollo de esta memoria.
    - "entorno". Contiene los archivos y directorios necesarios para levantar el entorno de ejecución del prototipo. Se basa en el uso de Docker, concretamente de Docker-Compose, para levantar los distintos servicios usando el archivo docker-compose.yml . Para la correcta configuración de cada servicio se hace uso de distintos volúmenes.
        - "apps". Directorio temporal creado por los scripts que contienen los drivers que se ejecutarán en el clúster de Spark
        - "data". Fuente de datos usada para el prototipo tanto para la Capa batch como para la capa de Stream
        - "scripts". Contiene un script para crear el topic en el Servicio de Kafka
        - "sql". Contiene el script de base de datos para crear la base de datos de PostgreSQL donde se guardan las vistas.

La implementación, que se encuentra en la carpeta "codigo", usa SBT como herramienta de construcción ya que la implementación del prototipo se ha realizado usando Scala como lenguaje de programación, ya que es el lenguaje nativo de Spark y por tanto rinde mejor que otros lenguajes disponibles para Spark y suele tener las mejoras de Spark suelen estar disponibles antes para este lenguaje.
Para la ejecución del prototipo se ha escogido el entorno basado en contenedores Docker por su facilidad para levantar un entorno local y su amplio uso. El entorno se compone un clúster Spark, limitado al nodo master y un nodo worker para no consumir muchos recursos, una

base de datos PostgreSQL, y un clúster Kafka, limitado al Apache Zookeper que gestiona el clúster y a un solo bróker.

4.1 Capa batch

El driver se compone de 2 transformaciones, una para la creación y etiquetado de trayectos y otra para la creación del evento de aceleración excesiva. Ambos leen archivos json con los frames de información transmitidos por los dispositivos y almacenados en un sistema de archivos distribuido.

4.1.1 Creación de trayectos

El código de la creación de los trayectos se puede encontrar en https://github.com/pmachiouam/MasterBigData_PF/blob/master/codigo/spark_proj/src/main/scala/org/uam/masterbigdata/JourneysHelper.scala.

1. Detección del cambio de valor de "ignition.state" para saber dónde comienza y donde termina el grupo de frames que forma un trayecto. Corresponde a la función setIgnitionStateChange.
    - Usamos una función de ventana que crea una columna con el valor 1 si hay un cambio de estado y 0 si no lo hay.
2. Quitar los frames con "ignition.state" no activo ya que un trayecto se compone de frames con la ignición activa.
3. Agrupar en trayectos. Corresponde a la función setGroupOfStateChangesToFrames
    - En base al punto 1, usamos una función de ventana para sumar la columna con los valores 1 y 0. La función suma los valores de la columna de forma que se crea un identificador incremental por cada agrupación. Todos los frames del mismo grupo tienen el mismo valor en esa columna
4. Establecer los valores iniciales del grupo de frames. Corresponde a la función setInitialStateChangeValues. Usando una función de ventana para saber el valor inicial dentro del grupo.
5. Establecer los valores finales del grupo de frames. Corresponde a la función setFinalStateChangeValues. Usando una función de ventana para saber el valor final dentro del grupo.
6. Determinar las diferencias de consumo y distancia entre los frames del grupo. Corresponde a la función setCountersValues. Se usa una función de ventana para comparar el valor actual con el anterior dentro del grupo.
7. Agregar los valores de grupo. Corresponde a la función aggregateStateChangeValues
    - Mínimo valor inicial coordenadas
    - Máximo valor final coordenadas
    - Suma de las diferencias de consumo y distancia para obtener el total del grupo.

4.1.2 Etiquetado de trayectos

El etiquetado de los trayectos se hace mediante algoritmos de Machine Learning disponibles en Spark gracias a librería MLLib. El código de la prueba de concepto se encuentra en https://github.com/pmachiouam/MasterBigData_PF/blob/master/codigo/spark_proj/src/main/scala/org/uam/masterbigdata/ClassifierHelper.scala

Para etiquetar los trayectos nos basamos en las coordenadas iniciales y finales del trayecto. Se ha probó inicialmente con un modelo de *regresión logística* que resulto ser suficiente para distinguir 6 tipos de trayectos distintos. Se considera que son trayectos del mismo tipo si las coordenadas no se desvían más de un 0.001 de la latitud o longitud ya que corresponde a una desviación de unos 111.1 metros (*http://wiki.gis.com/wiki/index.php/Decimal_degrees*).
Como no disponía de muchos ejemplos para entrenar el modelo he creado una pequeña función de data augmentation que genera nuevos trayectos a partir de otros con una modificación no mayor a 0.001 de la latitud y la longitud. La función genera 200 nuevos trayectos por cada uno de los trayectos que hemos usado como ejemplo. En la prueba unitaria de la función se han usado 6 trayectos como ejemplo de entrada y genera unos 1200 nuevos trayectos cuyas latitudes y longitudes de tienen una desviación estándar no mayor a 0.001 que corresponderían a una desviación no mayor a 111.1 metros respecto de la original. En la prueba unitaria, además de usar las desviaciones estándares para comprobar el correcto funcionamiento, es posible consultar la media, valor máximo y mínimo de las latitudes y longitudes de los nuevos datos generados.

Para el modelo se ha seleccionado *Regresión Logística* porque es uno de los modelos más sencillos y los resultados ofrecidos son más que satisfactorios (entorno al 0.99 para métricas de recall, precision, f1 y accuracy).
Para el entrenamiento se usa un pipeline cuyos pasos son:

- Indexar las etiquetas para pasar de texto a número
- Generación del vector de características

Al pipleline le hemos aplicado CrossValidation con los parámetros:

- RegParam: 0.1, 0.01, 0.001.
- MaxIter: 10, 20, 30.

Siendo los resultados muy similares para todas las combinaciones.
Finalmente hemos guardado el modelo para poder aplicarlo en el driver de creación de trayectos.

Cuando aplicamos el modelo, si no encontramos clases con una probabilidad superior a 0.9 etiquetamos el trayecto como desconocido.

Las distintas 6 etiquetas utilizadas son:

- “Tres Cantos – Moralzarzal". Que corresponde a los campos y valores
  - “start_location_latitude”: 40.614105
  - “start_location_longitude”: -3.711923
  - “end_location_latitude”: 40.6804
  - “end_location_longitude”: -3.976797
- “Moralzarzal - Tres Cantos". Que corresponde a los campos y valores
  - “start_location_latitude”: 40.6804
  - “start_location_longitude”: -3.976797
  - “end_location_latitude”: 40.614105
  - “end_location_longitude”: -3.711923
- “Tres Cantos - Arganzuela". Que corresponde a los campos y valores
  - “start_location_latitude”: 40.614334
  - “start_location_longitude”: -3.723002
  - “end_location_latitude”: 40.38668
  - “end_location_longitude”: -3.688833

- “Arganzuela - Tres Cantos". Que corresponde a los campos y valores
  - “start_location_latitude”: 40.38668
  - “start_location_longitude”: -3.688833
  - “end_location_latitude”: 40.614334
  - “end_location_longitude”: -3.723002
- “TC1 - TC2". Que corresponde a los campos y valores
  - “start_location_latitude”: 40.614448
  - “start_location_longitude”: -3.722875
  - “end_location_latitude”: 40.600124
  - “end_location_longitude”: -3.700752
- “TC2 - TC1". Que corresponde a los campos y valores
  - “start_location_latitude”: 40.600124
  - “start_location_longitude”: -3.700752
  - “end_location_latitude”: 40.614448
  - “end_location_longitude”: -3.722875

Hay una séptima etiqueta, “unknown” que como se indicó anteriormente, es utilizada para marcar los trayectos a los que no se les ha podido asignar una de las etiquetas anteriores con una probabilidad superior al 0.9.

4.1.3 Creación de evento de exceso de aceleración

Es un evento muy sencillo, si el valor del campo “can.vehicle.pedals.throttle.level” es igual o superior a 20, se crea el evento usando los valores del frames. El código se encuentra en https://github.com/pmachiouam/MasterBigData_PF/blob/master/codigo/spark_proj/src/main/scala/org/uam/masterbigdata/EventsHelper.scala, concretamente en la función createExcessiveThrottleEvent.

4.2 Capa de tiempo real

El driver de la capa de tiempo real define un pipeline de Spark Structured Streaming que consume los frames Kafka y guarda los resultados del procesamiento en PostgreSQL.

4.2.1 Creación de evento de descenso brusco del nivel de combustible.

Por vehículo debemos comprobar que si el nivel de combustible, campo “can.fuel.level” ha bajado en los últimos 5 minutos, campo “timestamp”. Es necesario mantener el estado de los frames ser capaces de detectar la bajada del nivel de combustible dentro del rango de tiempo. Spark Structured Streaming ofrece herramientas para mantener estado de un stream. El proceso queda definido en la función createFuelStealingEvent.
Para esta comprobación se hace uso de agrupación dispositivo, función “groupByKey” y estados mantenidos por Spark mediante la función “flatMapGroupWithState”. Lo que hacemos es guardar los estados relativos a los frames correspondientes a los últimos 5 minutos y comprobamos respecto al actual si la diferencia es superior al 5 %. Si es superior se crea el evento. Los estados más antiguos a 5 minutos son descartados.
Para la gestión de los estados es necesario definir tres clases:

- FuelStealingEventData. Que define el tipo de dato de entrada para la comprobación del cambio de estado.
- FuelStealingEventState. Define el estado

- FuelStealingEventResponse . Define la respuesta a la comprobación.

Definimos una función que recibe como valores de entrada, el valor por el que se agrupan los estados, una secuencia de datos de entrada y una secuencia de salidas. Dentro de la función por cada dato de entrada se recupera o crea el listado de estado si no existe, se comprueba si hay un decremento del nivel de combustible en los últimos 5 minutos y se devuelva el resultado, actualizando el estado.

Para aplicar la lógica se usan Datasets en lugar de Dataframes y hay que agrupar por el identificador del dispositivo y aplicar flatMapGroupsWithState para que los frames del dispositivo se comparen contra el estado guardado

5 Integración, pruebas y resultados

5.1 Entorno

Los drivers de Spark se han escrito en Scala y se usando Java 11, SBT 1.8 y Scala 2.12

Para el entorno de ejecución se ha usado Docker-Compose, con los servicios:

- Clúster de spark
  - Se han usado 2 volúmenes
    - Uno (apps) para compartir el archivo jar con los drivers y poder enviarlos al clúster usando Spark-submit
    - El otro (data) con el archivo json que contiene los frames y con el modelo de ML guardado
- PostgreSQL. Se ha creado un volumen para compartir el script de creación de base de datos.
- Kafka + Zookeeper. Se ha cerrado un volumen con los mismos datos usados en el clúster de spark para que el productor de consola del Kafka los publique.

El fichero de docker-compose.yml utilizado y la estructura de directorios en la que se apoya se puede encontrar en https://github.com/pmachiouam/MasterBigData_PF/tree/master/entorno

5.2 Pruebas

Para las pruebas se ha seleccionado un conjunto de datos pequeño pero representativo, que contiene información suficiente como para crear los tres casos de uso del prototipo y que resulte fácil de comprobar. El conjunto de datos se puede encontrar en https://github.com/pmachiouam/MasterBigData_PF/blob/master/entorno/data/datos.json
Este conjunto de datos es leído como fichero en la capa batch para realizar los casos de uso creación y clasificación de trayectos, y para la creación de eventos por aceleración excesiva. El mismo conjunto de datos es enviado a Kafka para su consumo por parte de la capa de tiempo real para ocuparse del caso de uso de la creación de eventos de descenso brusco del nivel de combustible.

5.2.1 Datos usados

El set de datos se compone de 20 frames con información suficiente como para crear:

- 4 trayectos
  - Primer trayecto corresponde a los frames 2,3 y 4 que van de las 9:55:27 del 2 de Febrero de 2023 a las 9:57:27 de la misma fecha.
  - Segundo trayecto corresponde a los frames 7,8 y 9 que van de las 10:55:27 del 2 de Febrero de 2023 a las 10:57:27 de la misma fecha.
  - Tercer trayecto corresponde a los frames 12,13 y 14 que van de las 11:55:27 del 2 de Febrero de 2023 a las 11:57:27 de la misma fecha.
  - Cuarto trayecto corresponde a los frames 17,18 y 19 que van de las 12:55:27 del 2 de Febrero de 2023 a las 13:01:27 de la misma fecha.
- 2 eventos de exceso de aceleración.
  - A las 11:56:27 del 2 de Febrero de 2023, frame 13, y a las 12:58:27 del 2 de Febrero de 2023, frame 18
- 2 eventos de descenso brusco de nivel de combustible

- A las 13:01:27 del 2 de Febrero de 2023, frame 18, y a las 13:04:27 del 2 de Febrero de 2023, frame 19.

5.3 Resultados

Los resultados obtenidos se pueden consultar en el anexo 6.3.4

5.3.1 Creación de trayectos y clasificación

Los 4 trayectos son creados correctamente, correspondiendo a las agrupaciones de frames con el campo "ingition.status" igual a true entre frames con el campo "ingition.status" igual a false, como se indicaba en el apartado anterior.
El etiquetado también es correcto, 2 trayectos son etiquetados como "unknown" porque no se le ha podido asignar ninguna de las 6 etiquetas definidas con una probabilidad superior a 0,9, esto se debe a que las latitudes y las longitudes de inicio y fin difieren en más de 0.001 respecto de las de las etiquetas.
Los 2 trayectos que está bien etiquetados podemos comprobar como las latitudes y longitudes corresponde exactamente con las de las etiquetas.
Se pueden ver en el anexo 6.3.4.1

5.3.1.1 Entrenamiento del clasificador.

Usamos una regresión logística

Para el entrenamiento se han usado 1200 muestras generadas con la función de aumento de dato. Concretamente se han usado 355 para test y 845 para entrenamiento. Las métricas obtenidas en test son:

- F1: 1.0
- Precision: 1.0
- Recall: 1.0
- Accuracy: 1.0

Métricas sospechosamente buenas, pero hay que tener en cuenta que realmente no se ha metido ruido en los datos de entreno sino variaciones de lo que se consideraban valores correctos para las latitudes y longitudes (de inicio y fin), y los valores de las latitudes y longitudes (de inicio y fin) no son próximos entre sí, facilitando su clasificación.

5.3.2 Creación de evento de aceleración excesiva

La creación de los eventos ha sido satisfactoria ya que ha encontrado los dos frames que contienen un valor del campo "can.vehicle.pedal.throttle" igual o superior al 21% que son las de los frames 13 y 18 que corresponden a 11:56:27 y 12:58:27 del 2 de Febrero de 2023, correspondientemente. En el anexo 6.3.4.2 corresponde a los eventos de con el campo "type_id" igual 1.

5.3.3 Creación de evento de evento de descenso brusco del nivel de combustible

La creación de ambos eventos también ha sido correcta, en este caso, el sistema ha sido capaz de comprobar un descenso del nivel de combustible observando la diferencia del campo "can.fuel.level" entre distintos frames cuyo rango de tiempo no sea superior a 5 minutos. Rango de tiempo correspondiente a los frames, no al procesamiento. En el anexo 6.3.4.2 corresponde a los eventos de con el campo "type_id" igual 2.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Spark ofrece un entorno ideal para el desarrollo de una arquitectura Lambda.
La abstracción que ofrece para múltiples fuentes y destinos, tanto de la capa Batch como de la capa RealTime, evitan que los desarrolladores empleen tiempo innecesario en su implementación y se focalicen en la lógica de transformaciones y acciones sobre los datos.
El desarrollo estás lógicas también se ve favorecido en el entorno de Spark porque en un mismo proyecto podemos tener distintas lógicas fomentando la reutilización de código, así como una mejor comprensión de estás

6.2 Trabajo futuro

Aunque en el trabajo se ha tratado la arquitectura Lambda, el uso de Spark se ha centrado en las capas Batch y RealTime para generar las vistas que se consultan desde la capa de Servicio. En el trabajo no me he centrado en esta capa, aunque Spark ofrece un módulo de consulta SQL porque existen herramientas de BigData que permiten hacer consultan sobre los datos de forma más eficiente que Spark, como es el caso de Cassandra, una base datos no relacional distribuida que escala muy bien y que permite una eficiencia muy alta en las consultas siempre que estas tengan un claro caso de uso, no es para consultas generalistas sobre los datos. El siguiente paso sería usar Cassandra como capa de servicio para guardar las vistas y así cerrar la arquitectura.

7 Referencias

[1] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys Tutorials, 17(4) Pag:2347-2376. https://ieeexplore.ieee.org/document/7123563

[2] Keyur K Patel, Sunil Patel, Carlos Salazar. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. IJESC Volumen 6 Issue 5. Pag:6122-6131 https://www.researchgate.net/publication/330425585

[3] Singh, Kamakhya & Behera, Rajat & Mantri, Jibendu. (2019). Big Data Ecosystem: Review on Architectural Evolution: Proceedings of IEMIS 2018, Volume 2 https://www.researchgate.net/publication/327389927_Big_Data_Ecosystem_Review_on_Architectural_Evolution_Proceedings_of_IEMIS_2018_Volume_2

[4] Nathan Marz. (2015) Big Data: Principles and best practices of scalable realtime data systems. Manning Publications. Pag: 25-61

[5] Martin Feick, Niko Kleer, and Marek Kohn (2018): SKILL 2018, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 https://dl.gi.de/bitstream/handle/20.500.12116/28983/SKILL2018-05.pdf?sequence=1&isAllowed=y

[6] Ian L.Dryden, David J Hodge. (2018). Estimating Vehicle Fuel Consumption and Emissions using GPS BigData. Int. J. Environ. Res. Public Health 2018, 15(4), 566.

[7] Ali, Muhammad and Kamal, Muhammad D. and Tahir, Ali and Atif, Salman. (2021). Fuel Consumption Monitoring through COPERT Model—A Case Study for Urban Sustainability. Pag:13-21

[8] Ian L.Dryden, David J Hodge, (2018), Journeys in big data statistics, Statistics & Probability letters, Volume 136, May 2018. Pag 121-125.

[9] C. Prehofer and S. Mehmood (2020), "Big Data Architectures for Vehicle Data Analysis," *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA. Pag: 3404-3412

[10] Tyler Akidau, Slava Chernyak, R. L. (2018). Streaming Systems. O'Reilly Media, Inc. Pag:19-76

8 Apéndice

8.1 Modelos de datos

8.1.1 Frame de información de los dispositivos IoT

Corresponde con un archivo JSON, donde se pueden encontrar distintos niveles de profundidad por atributo según la complejidad. Se compone de los siguientes campos:

- "id". Identificador único del
- "version". Versión del "modelo de datos" o "protocolo". Identificador del esquema usado para los datos. Cada versión puede tener más o menos campos.
- "timestamp". Fecha en la que el dispositivo genero el frame de información.
- "server". Información de registro la recepción del frame. Se espera que crezca
    - "timestamp". Fecha en la que se recibió el frame y se estableció el "id" y "version".
- "attributes". Atributos del dispositivo
    - "tenantId". Identificador del propietario del dispositivo.
    - "deviceId". Identificador del dispositivo
    - "manufacturer". Nombre del fabricante del dispositivo
    - "model". Modelo del dispositivo
    - "identifier". Número del modem del dispositivo.
- "device". Sensores del propio dispositivo
    - "battery". Batería del dispositivo
        - "voltage". Lectura del voltaje
        - "level". Nivel de batería. De 0 a 100.
    - "mileage". Odómetro mantenido por el propio dispositivo.
        - "distance". Distancia calculada por del dispositivo en base al GPS
- "can". Sensores del propio vehículo que se acceden mediante el puerto OBDII o Canbus
    - "vehicle". Sensores del vehículo que no son ni del motor, ni de la batería y ni relacionados con el consumo.
        - "mileage". Odómetro mantenido por el vehículo
            - "distance". Distancia total recorrida en metros.
        - "pedal". Sensores sobre los pedales del vehículo
            - "throttle". Acelerador
                - "level". Porcentaje pisado del acelerador
        - "cruise". Control de crucero, para circular a una velocidad determinada
            - "status". Activo o no
        - "handBrake". Freno de mano
            - "status". Activo o no
        - "doors". Puertas del vehículo.
            - "indicator". Notificación de puerta abierta
                - "status". Activo o no
        - "lights". Luces del vehículo
            - "hazard". Luces de emergencia
                - "status". Activo o no

                - “fog”. Luces de antiniebla
                    - “status”. Activo o no
            - “seatBelts”. Cinturones de seguridad
                - “indicator”. Notificación de puerta abierta
                    - “status”. Activo o no
            - “beams”. Focos
                - “high”. Indicador de si lo focos están alto
                    - “status”. Activo o no
            - “lock”. Seguro de las puertas
                - “central”. Cierre centralizado de los seguros
                    - “status”. Activo o no
            - “gear”. Marchas
                - “reverse”. Marcha atrás
                    - “status”. Activo o no
            - “airConditioning”. Aire acondicionado
                - “status”. Activo o no
        - “engine”. Sensores del motor.
            - “time”. Tiempo de actividad del motor
                - “duration”. Duración total del motor activo
        - “battery”. Sensores de la batería.
            - “charging”. Si se encuentra la batería recargándose.
                - “status”. Si o no
        - “fuel”. Sensores relacionados con el consumo de combustible.
            - “consumed”. Consumo de combustible.
                - “volume”. Volumen total consumido. En litros.
            - “level”. Nivel del tanque de combustible.
- “gnss”. Sensores relacionados con la geolocalización del dispositivo.
    - “type”: Que se usó para calcular.
        - GPS
        - GSM, triangulación de antenas GSM
    - “coordinate”
        - “lat”. Latitud. Grados
        - “lng”. Longitud. Grados
    - “altitude”. Altura en metros
    - “speed”. Velocidad en Km/h
    - “course”. Dirección en grados (0 a 360)
    - “address”. Dirección, si se aplicó geolocalización inversa.
    - “satellites”. Numero de satélites utilizados para el cálculo. Caso de GPS
- “ignition”. Sensor de la llave de ignición de vehículo.
    - “status”. Activa o no

8.1.2 Trayectos

Los trayectos son agregaciones de consumo y distancia de un dispositivo asociado a un vehículo entre unas coordenadas de inicio y otras de fin. Estas agregaciones se realizan sobre un conjunto de frames consecutivos en el tiempo que mantienen el estado “ignition.status” activo de manera continua. Queda definido por los campos:

- “id”. UUID para identificar unívocamente cada trayecto. Se genera al crear el trayecto
- “device_id”. Identificador del dispositivo

- “start_timestamp”. Valor del campo “timestamp” inicial, que corresponde al primer frame con el campo “ignition.status” activo, el anterior estaban inactivos.
- “start_location_address”. Valor del campo “gnss.address” inicial, que corresponde al primer frame con el campo “ignition.status” activo. Los anteriores estaban inactivos.
- “start_location_latitude”. Valor del campo “gnss.coordinate.lat” inicial, que corresponde al primer frame con el campo “ignition.status” activo. Los anteriores estaban inactivos.
- “start_location_longitude”. Valor del campo “gnss.coordinate.longitude” inicial, que corresponde al primer frame con el campo “ignition.status” activo. Los anteriores estaban inactivos.
- “end_timestamp”. Valor del campo “timestamp” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
- “end_location_address”. Valor del campo “gnss.address” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
- “end_location_latitude”. Valor del campo “gnss.coordinate.lat” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
- “end_location_longitude”. Valor del campo “gnss.coordnate.lng” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
- “distance”. Diferencia entre el último y el primer valor del campo “can.vehicle.mileage.distance”
- “consumption”. Diferencia entre el último y el primer valor del campo “can.fuel.consumed.volume”
- “label”. Etiqueta con la clase del trayecto. Si no se conoce se marcará como “unknown”

8.1.3 Eventos

Son registros de valores de los sensores que se consideran importantes, ya sea porque se han superado un umbral o su evolución en el tiempo es anómala. Se componen de los siguientes campos:

- “id”. UUID para identificar unívocamente cada evento. Se genera al crear el evento.
- “created”. Corresponde con el campo “timestamp” del frame que disparó el evento.
- “type_id”. Identifica el tipo de evento. En este caso tenemos:
    - 1 para eventos correspondientes a pisar el pedal del acelerador, campo “can.vehicle.pedals.throttle.level”, por encima del 20%.
    - 2 para eventos correspondientes a un descenso del 5% del nivel de combustible en menos de 5 minutos
- “location_address”. Corresponde con el campo “gnss.address” del frame que disparó el evento.
- “location_latitude”. Corresponde con el campo “gnss.coordinate.lat” del frame que disparó el evento.
- “location_longitude”. Corresponde con el campo “gnss.coordinate.lng” del frame que disparó el evento.
- “value”. Valor descriptivo del evento.
    - Para eventos de type_id = 1 indicamos el porcentaje de pedal
    - Para eventos de type_id = 2 indicamos que el porcentaje del nivel combustible descendió más de 5% en 5 minutos.

8.1.4 Resultados

8.1.4.1 Trayectos.

Los trayectos obtenidos tras ejecutar la capa batch y obtenidos como jsons en la aplicación web.
Url utilizada. http://localhost:8080/api/v1.0/trayectos/763738942838145024


```
[
  {
    "id": "1bea3f72-e85b-4665-9aa9-e634f93799c9",
    "device_id": "763738942838145024",
    "start_timestamp": "1675673727000",
    "start_location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid, 28760, España",
    "start_location_latitude": "40.605957",
    "start_location_longitude": "-3.711923",
    "end_timestamp": "1675673847000",
    "end_location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid, 28760, España",
    "end_location_latitude": "40.60596",
    "end_location_longitude": "-3.711921",
    "distance": "2000",
    "consumption": "200",
    "label": "unknown"
  },
  {
    "id": "db623add-3d3a-4e62-87bf-9ea74c84f5da",
    "device_id": "763738942838145024",
    "start_timestamp": "1675677327000",
    "start_location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid, 28760, España",
    "start_location_latitude": "40.614105",
    "start_location_longitude": "-3.711923",
    "end_timestamp": "1675677447000",
    "end_location_address": "Moralzarzal",
    "end_location_latitude": "40.6804",
    "end_location_longitude": "-3.976797",
    "distance": "4000",
    "consumption": "600",
    "label": "Tres Cantos - Moralzarzal"
  },
  {
    "id": "c53c2b38-efe8-4830-a4c9-c6431ab6676a",
    "device_id": "763738942838145024",
    "start_timestamp": "1675680927000",
    "start_location_address": "Moralzarzal",
    "start_location_latitude": "40.6804",
    "start_location_longitude": "-3.976797",
    "end_timestamp": "1675681047000",
    "end_location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid,
```

```
28760, España",
    "end_location_latitude": "40.614105",
    "end_location_longitude": "-3.711923",
    "distance": "6000",
    "consumption": "800",
    "label": "Moralzarzal - Tres Cantos"
  },
  {
    "id": "a165c25b-04df-4619-acd7-415f5b04b3df",
    "device_id": "763738942838145024",
    "start_timestamp": "1675684527000",
    "start_location_address": " Tres Cantos",
    "start_location_latitude": "40.614105",
    "start_location_longitude": "-3.723002",
    "end_timestamp": "1675684887000",
    "end_location_address": "Arganzuela",
    "end_location_latitude": "40.38678",
    "end_location_longitude": "-3.688923",
    "distance": "8000",
    "consumption": "900",
    "label": "unknown"
  }
]
```


8.1.4.2 Eventos

Los trayectos obtenidos tras ejecutar la capa batch y obtenidos como jsons en la aplicación web.

http://localhost:8080/api/v1.0/eventos/763738942838145024


```
[
  {
    "id": "dc50278c-5103-4a24-b5c4-2a9b00caceb6",
    "device_id": "763738942838145024",
    "created": "1675680987000",
    "type_id": "1",
    "location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid, 28760, España",
    "location_latitude": "40.605957",
    "location_longitude": "-3.711922",
    "value": "25%"
  },
  {
    "id": "39efcf55-9974-4832-909d-2db2014fb527",
    "device_id": "763738942838145024",
    "created": "1675684707000",
    "type_id": "1",
    "location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid, 28760, España",
    "location_latitude": "40.605957",
```

```
  "location_longitude": "-3.711922",
  "value": "21%"
 },
 {
  "id": "7debb028-df42-40aa-8b91-6790002dee34",
  "device_id": "763738942838145024",
  "created": "1675684887000",
  "type_id": "2",
  "location_address": "Arganzuela",
  "location_latitude": "40.38678",
  "location_longitude": "-3.688923",
  "value": "5% in less or 5 minutes"
 },
 {
  "id": "bfae448c-2115-4670-ad10-7f6ce6a31120",
  "device_id": "763738942838145024",
  "created": "1675685067000",
  "type_id": "2",
  "location_address": "Avenida de la Vega, Tres Cantos, Comunidad de Madrid, 28760, España",
  "location_latitude": "40.605957",
  "location_longitude": "-3.711923",
  "value": "5% in less or 5 minutes"
 }
]
```


8.2 Script de creación de base de datos.

Script de creación de la base de datos de PostgreSQL para guardar las vistas de las capas de batch y tiempo real. Se encuentra en https://github.com/pmachiouam/MasterBigData_PF/tree/develop/entorno/sql/db.sql

```
--
-- PostgreSQL database dump
--


-- Dumped from database version 11.1 (Debian 11.1-1.pgdg90+1)
-- Dumped by pg_dump version 11.1 (Debian 11.1-1.pgdg90+1)


SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET client_min_messages = warning;
SET row_security = off;
SET default_tablespace = '';
SET default_with_oids = false;
```

```
--Setup database
DROP DATABASE IF EXISTS tracking;
CREATE DATABASE tracking;
\c tracking;


\echo 'Creando la base de datos '
CREATE TABLE public.journeys (
id uuid NOT NULL,
device_id int8 NOT NULL,
start_timestamp timestamp NOT NULL,
start_location_address text NOT NULL,
start_location_latitude float NOT NULL,
start_location_longitude float NOT NULL,
end_timestamp timestamp NULL,
end_location_address text NULL,
end_location_latitude float NULL,
end_location_longitude float NULL,
distance int8 NULL,
consumption int8 NULL,
label text NULL
);
comment on column journeys.id is 'Identificador del trayecto';
comment on column journeys.device_id is 'Identificador del dispositivo que realiza el trayecto';
comment on column journeys.start_timestamp is 'Fecha de inicio del trayecto';
comment on column journeys.start_location_address is 'Dirección de la localización del inicio del trayecto';
comment on column journeys.start_location_latitude is 'Latitud de la localización del inicio del trayecto';
comment on column journeys.start_location_longitude is 'Longitud de la localización del inicio de trayecto';
comment on column journeys.end_timestamp is 'Fecha de fin de trayecto. Solo se establece si está terminado el trayecto';
comment on column journeys.end_location_address is 'Dirección de la localización del final de trayecto. Solo se establece si está terminado el trayecto';
comment on column journeys.end_location_latitude is 'Latitud de la localización del final de trayecto. Solo se establece si está terminado el trayecto';
comment on column journeys.end_location_longitude is 'Longitud de la localización del final de trayecto. Solo se establece si está terminado el trayecto';
comment on column journeys.distance is 'Distancia recorrida en metros. Solo se establece si está terminado el trayecto';
comment on column journeys.consumption is 'Consumo realizado en trayecto en litros. Solo se establece si está terminado el trayecto';
comment on column journeys.label is 'Etiqueta asociada al trayecto que permite identificar trayectos con el mismo inicio y fin (location_latitude, location_longitude)';


CREATE TABLE public.frames (
id int8 NOT NULL,
device_id int8 NOT NULL,
created timestamp NOT NULL,
received timestamp NOT NULL,
```

```
location_created timestamp NULL,
location_address text NULL,
location_latitude numeric(11, 7) NULL,
location_longitude numeric(11, 7) NULL,
location_altitude float8 NULL,
location_speed float4 NULL,
location_valid bool NULL,
location_course float4 NULL,
ignition bool NULL
--Pendiente meter las entradas de sondas de combustible,
);
comment on column frames.id is 'Indetificador único del frame';
comment on column frames.device_id is 'Identificador del dispositivo';
comment on column frames.created is 'Fecha de creción de la trama';
comment on column frames.received is 'Fecha de recepción de la trama en el sistema';
comment on column frames.location_created is 'Fecha de creación de la localización del dispositivo';
comment on column frames.location_address is 'Dirección de la localización del dispositivo';
comment on column frames.location_latitude is 'Latitud de la localización del dispositivo';
comment on column frames.location_longitude is 'Longitud de la localización del dispositivo';
comment on column frames.location_altitude is 'Altitud de la localización del dispositivo';
comment on column frames.location_speed is 'Velocidad de la localización del dispositivo';
comment on column frames.location_valid is 'Si o no valida la localización del dispositivo';
comment on column frames.location_course is 'Dirección en grados de la localización del dispositivo';
comment on column frames.ignition is 'Si la trama contiene el estado de la llave de contacto del vehículo';


CREATE TABLE public.events (
id uuid NOT NULL,
device_id int8 NOT NULL,
created timestamp NOT NULL,
type_id int8 NOT NULL,
location_address text NULL,
location_latitude numeric(11, 7) NULL,
location_longitude numeric(11, 7) NULL,
value text NULL
);
comment on column events.id is 'Indetificador único del evento';
comment on column events.device_id is 'Identificador del dispositivo';
comment on column events.created is 'Fecha de creción de la del evento';
comment on column events.type_id is 'Identificador del tipo de evento';
comment on column events.location_address is 'Dirección de la localización del dispositivo que generó el evento';
comment on column events.location_latitude is 'Latitud de la localización del dispositivo que generó el evento';
comment on column events.location_longitude is 'Longitud de la localización del dispositivo que generó el evento';
comment on column events.value is 'Información extra sobre el evento';
```

8.3 Código

Repositorio:
https://github.com/pmachiouam/MasterBigData_PF/tree/develop