- **Soporte para diversos algoritmos.** Soporta una variedad de algoritmos de ML comunes, incluyendo regresión lineal, regresión logística, árboles de decisión, y redes neuronales, entre otros.

En la Figura 23, vemos un **flujo de funcionalidades** de Redshift ML.

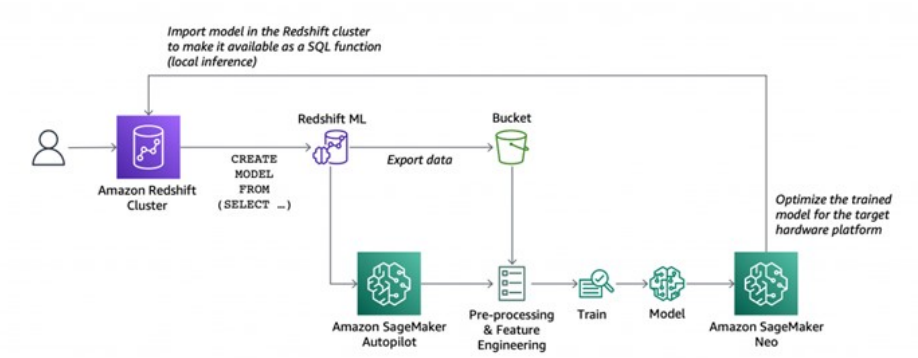


Figura 23. Flujo de funcionalidades de Redshift ML. Fuente: Poccia, 2021.

Las **ventajas** más destacables de Redshit ML son las siguientes:

- **Facilidad de uso.** Los analistas de datos pueden crear y utilizar modelos de ML sin salir de su entorno de SQL habitual, lo que reduce la barrera de entrada para la adopción de ML.
- **Integración directa.** La integración directa con Redshift elimina la necesidad de mover datos a sistemas externos para el entrenamiento y la inferencia, lo que reduce la latencia y los costos asociados a la transferencia de datos.
- **Escalabilidad.** Al utilizar Amazon SageMaker, Redshift ML se beneficia de la escalabilidad de SageMaker para manejar grandes volúmenes de datos y entrenar modelos complejos.
- **Automatización.** La automatización del proceso de entrenamiento y despliegue de modelos facilita la implementación de soluciones de ML sin la necesidad de gestionar la infraestructura subyacente.

- **Coste-efectividad.** Aprovecha el modelo de precios basado en uso de SageMaker, permitiendo a las organizaciones pagar solo por los recursos de entrenamiento e inferencia que utilizan.

Los siguientes son algunos ejemplos de **casos de uso** de Redshift ML:

- **Predicción de *churn* de clientes.** Las empresas pueden utilizar Redshift ML para predecir la tasa de abandono de clientes, identificando patrones en los datos históricos que indican un mayor riesgo de *churn*.
- **Detección de fraude.** Redshift ML puede entrenar modelos para detectar transacciones fraudulentas en tiempo real al analizar los patrones y las anomalías en los datos de transacciones.
- **Segmentación de clientes.** Permite crear modelos que segmenten a los clientes en diferentes grupos basados en sus comportamientos y características, facilitando campañas de *marketing* más efectivas.
- **Pronóstico de demanda.** Utiliza datos históricos de ventas y otros factores para predecir la demanda futura de productos, lo que ayuda a optimizar la gestión de inventarios.
- **Análisis de sentimiento.** Permite analizar comentarios de clientes y reseñas de productos para identificar el sentimiento general y las opiniones sobre productos o servicios.

Se describe, a continuación, un **ejemplo de uso de Redshift ML,** con los pasos a seguir:

- **Crear un modelo de ML.** Escribir una consulta SQL para crear un modelo de predicción de *churn*.

```
sql
CREATE MODEL customer_churn_model
FROM (
```

```
    SELECT
        customer_id,
        churned,
        age,
        tenure,
        spend
    FROM customer_data
)
TARGET churned
FUNCTION predict_churn;
```

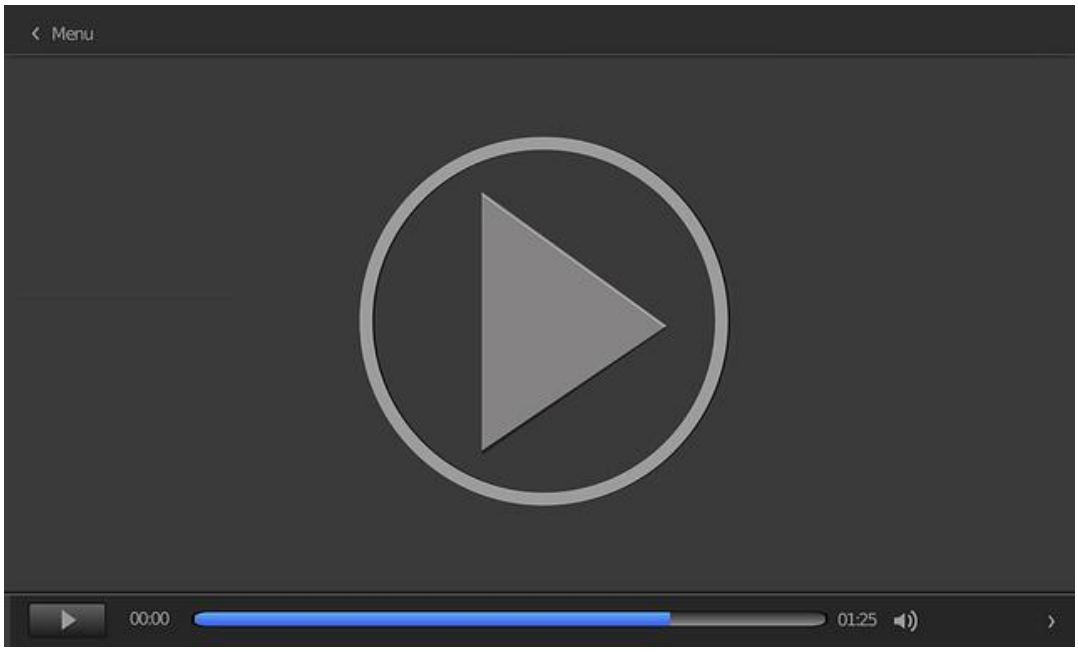
- **Entrenamiento del modelo.** Redshift ML utiliza Amazon SageMaker para entrenar el modelo automáticamente.

- **Usar el modelo para inferencia.** Una vez entrenado, realiza predicciones directamente en SQL.

```
sql
SELECT customer_id, predict_churn(age, tenure, spend)
FROM new_customer_data;
```

Amazon Redshift ML facilita la integración de ML en el análisis de datos, por lo que permite que las organizaciones aprovechen las capacidades de ML directamente en su *data warehouse*. Con su enfoque en la facilidad de uso, escalabilidad y costo-efectividad, Redshift ML empodera a los analistas de datos para implementar soluciones de ML sin necesidad de habilidades avanzadas en ML, lo que acelera el tiempo de obtención de *insights* valiosos.

A continuación, se muestra el vídeo *AWS re:Invent 2020: Introducing Amazon Redshift Machine Learning* (AWS Events, 2021), que profundiza en las capacidades del servicio de AWS Redshift ML.



AWS re:Invent 2020: Introducing Amazon Redshift Machine Learning.

Accede al vídeo:

https://www.youtube.com/embed/3BN1w8JUtD4

Integración con Spark

La **integración de AWS Redshift ML con Apache Spark** permite aprovechar las capacidades avanzadas de procesamiento y análisis de datos de Spark junto con las funcionalidades de ML de Redshift ML. Esta combinación es potente para escenarios donde se requieren análisis complejos y volúmenes masivos de datos que Spark puede manejar de manera eficiente antes de enviar datos a Redshift para el entrenamiento y las inferencias de ML.

Esta integración simplifica y acelera las **aplicaciones** Apache Spark a través del acceso a los datos de Amazon Redshift de los servicios de análisis de AWS, como Amazon EMR, AWS Glue y Amazon SageMaker. Con Amazon EMR, AWS Glue y SageMaker, se pueden crear con rapidez aplicaciones de Apache Spark, que leen desde su almacenamiento de datos de Amazon Redshift y que escriben en este, sin comprometer el rendimiento ni la coherencia transaccional.

La integración entre ambos también utiliza **credenciales** basadas en AWS IAM para mejorar la seguridad. Con la integración de Amazon Redshift para Apache Spark, no hay configuración ni mantenimiento manual de versiones no certificadas de conectores de terceros. Puedes comenzar con los trabajos de Apache Spark utilizando los datos en Amazon Redshift en segundos. Esta nueva integración mejora el rendimiento de las aplicaciones Apache Spark a través del uso de los datos de Amazon Redshift.

El **objetivo** de la integración es utilizar Spark para procesar y transformar grandes volúmenes de datos y, luego, utilizar Redshift ML para entrenar modelos de ML y realizar inferencias directamente en el *data warehouse* de Redshift. Esto permite a las organizaciones utilizar una arquitectura de análisis de datos flexible y escalable.

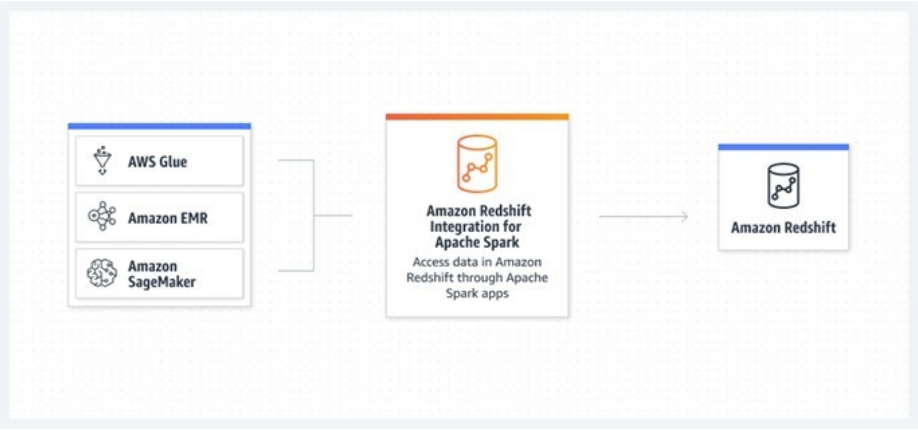


Figura 24. Diagrama de la integración. Fuente: Amazon Web Services, s. f.-f.

Entre las principales **funcionalidades** de esta integración, se enumeran las siguientes:

- **Extracción y transformación de datos con Spark.** Utiliza Spark para procesar y transformar datos a gran escala. Spark es muy eficiente en el procesamiento distribuido de grandes volúmenes de datos.
- **Carga de datos a Redshift.** Después de transformar los datos, se pueden cargar a Amazon Redshift para su almacenamiento y análisis adicional. Esto se hace generalmente usando el conector Spark-Redshift.
- **Entrenamiento de modelos con Redshift ML.** Utiliza los datos cargados en Redshift para crear y entrenar modelos de ML utilizando Redshift ML.
- **Inferencia directa desde Redshift.** Realiza predicciones y análisis de ML directamente en Redshift utilizando SQL.

Los **pasos** para esta integración serían los siguientes:

- **Configuración del entorno Spark.** Configura un clúster de Spark, ya sea en AWS EMR (Elastic MapReduce) o en otra infraestructura gestionada.
- **Procesa de datos con Spark.** Usa Spark para leer, limpiar, transformar y enriquecer los datos desde diversas fuentes. Por ejemplo, leer datos desde Amazon S3:

```
python
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("RedshiftMLIntegration").getOrCreate()
df = spark.read.csv("s3://my-bucket/my-data.csv", header=True,
inferSchema=True)
```

- **Carga de datos transformados a Redshift.** Utiliza el conector Spark-Redshift para cargar los datos procesados a Amazon Redshift:

```
python
df.write \
    .format("com.databricks.spark.redshift") \
    .option("url", "jdbc:redshift://redshift-cluster:5439/mydb") \
    .option("dbtable", "transformed_data") \
    .option("user", "myuser") \
    .option("password", "mypassword") \
    .option("aws_iam_role", "arn:aws:iam::account-
id:role/RedshiftCopyUnload") \
    .save()
```

Entre las **ventajas** de esta integración, podemos destacar lo siguiente:

- **Eficiencia en el procesamiento de datos.** Spark puede manejar grandes volúmenes de datos y realizar transformaciones complejas de manera eficiente antes de cargar los datos en Redshift.
- **Escalabilidad.** Tanto Spark como Redshift son altamente escalables, lo que permite manejar grandes cargas de trabajo y volúmenes de datos.
- **Simplificación del flujo de trabajo de ML.** La integración permite realizar todo el flujo de trabajo de ML, desde la preparación de datos hasta la inferencia, utilizando herramientas robustas y escalables.
- **Flexibilidad.** Utilizar Spark para la preparación de datos y Redshift para ML permite aprovechar lo mejor de ambas plataformas, lo que ofrece una solución flexible para diferentes necesidades analíticas.

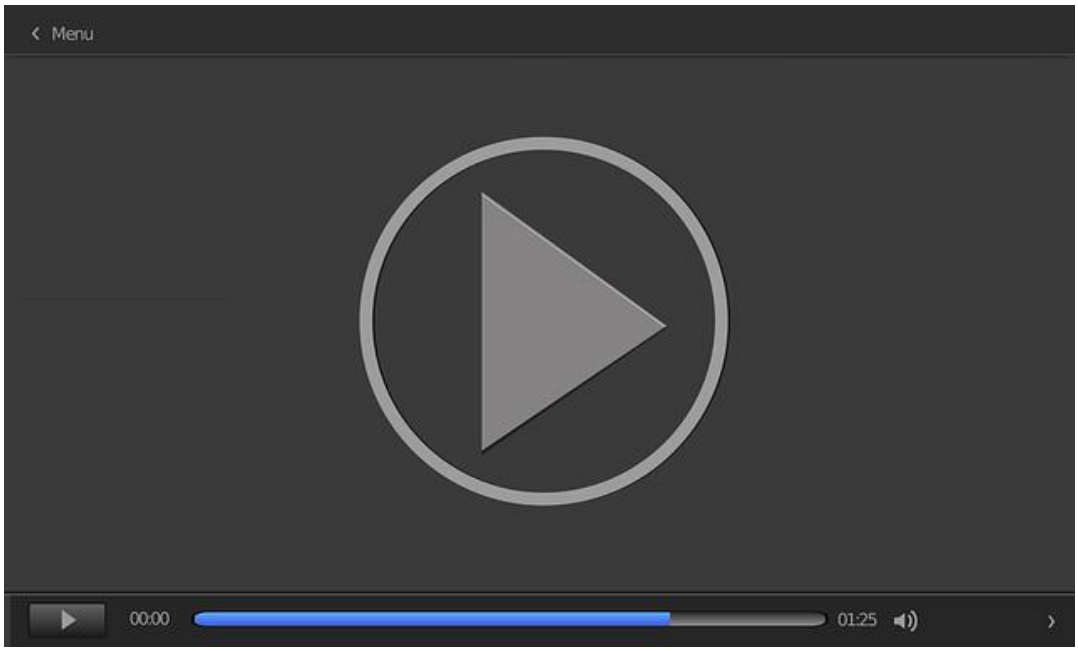
Los siguientes son algunos ejemplos de **casos de uso** de esta integración:

- **Análisis de *logs* y predicción de anomalías.** Procesar grandes volúmenes de *logs* con Spark, cargar los datos transformados a Redshift y usar Redshift ML para detectar anomalías en tiempo real.

- **Predicción de demanda.** Utilizar Spark para combinar y procesar datos de múltiples fuentes, cargar los datos a Redshift y entrenar modelos de predicción de demanda con Redshift ML.
- **Análisis de sentimiento en redes sociales.** Procesar datos de redes sociales con Spark para extraer características relevantes, cargar los datos en Redshift y usar modelos de ML para análisis de sentimiento.

La **integración de AWS Redshift ML con Apache Spark** proporciona una solución poderosa y flexible para manejar y analizar grandes volúmenes de datos. Al combinar la capacidad de procesamiento masivo de Spark con las capacidades de ML de Redshift ML, las organizaciones pueden implementar flujos de trabajo de análisis y ML de manera eficiente y escalable. Esta integración es ideal para escenarios donde la preparación de datos complejos y el análisis predictivo son críticos para la toma de decisiones empresariales.

Seguidamente, se muestra el vídeo *Amazon Redshift integration for Apache Spark - Demo | Amazon Web Services* (Amazon Web Services, 2022b), que es un demo de la integración de AWS Redshift y Spark.



Amazon Redshift integration for Apache Spark - Demo | Amazon Web Services.

Accede al vídeo:

https://www.youtube.com/embed/3Me8SAruNB0

3.5. Referencias bibliográficas

Amazon Web Services. (2020a, mayo 18). *Getting Started with Amazon Redshift - AWS Online Tech Talks* [Vídeo]. YouTube. https://youtu.be/dfo4J5ZhlKI

Amazon Web Services. (2020b, diciembre 9). *Amazon Redshift Data Sharing Workflow* [Vídeo]. YouTube. https://youtu.be/EXioFirlrnA

Amazon Web Services. (2020c, diciembre 9). *Amazon Redshift Data Sharing Use Cases* [Vídeo]. YouTube. https://youtu.be/sIoTB8B5nn4

Amazon Web Services. (2021a, noviembre 30). *Introduction to Data Warehousing on AWS with Amazon Redshift | Amazon Web Services* [Vídeo]. YouTube. https://youtu.be/lWwFJV_9PoE

Amazon Web Services. (2021b, septiembre 29). *An Overview of Amazon Redshift Query Editor V2 | Amazon Web Services* [Vídeo]. YouTube. https://youtu.be/IwZNIroJUnc

Amazon Web Services. (2022a, enero 10). *Amazon Redshift Serverless - End to End Use Case | Amazon Web Services* [Vídeo]. YouTube. https://youtu.be/gEAWSqjthXs

Amazon Web Services. (2022b, diciembre 19). *Amazon Redshift integration for Apache Spark - Demo | Amazon Web Services* [Vídeo]. YouTube. https://youtu.be/3Me8SAruNB0

Amazon Web Services. (s. f.-a). *Amazon S3.* https://aws.amazon.com/es/s3/

Amazon Web Services. (s. f.-b). *Operaciones por lotes de Amazon S3.* https://aws.amazon.com/es/s3/features/batch-operations/

Amazon Web Services. (s. f.-c). *Replicación de Amazon S3*. https://aws.amazon.com/es/s3/features/replication/

Amazon Web Services. (s. f.-d). *Cumplimiento de los requisitos de conformidad mediante el control de tiempo de replicación de S3 (S3 RTC)*. https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/replication-time-control.html

Amazon Web Services. (s. f.-e). *Amazon Redshift*. https://aws.amazon.com/es/redshift/

Amazon Web Services. (s. f.-f). *Integración de Amazon Redshift para Apache Spark.* https://aws.amazon.com/es/redshift/features/integration-for-apache-spark/

Amazon Web Services. (s. f.-g). *Amazon S3.* https://aws.amazon.com/es/s3/

Amazon Web Services. (s. f.-h). *Puntos de acceso de Amazon S3.* https://aws.amazon.com/es/s3/features/access-points/

Amazon Web Services. (s. f.-i). *Introducción a Amazon S3.* https://aws.amazon.com/es/s3/getting-started/

Amazon Web Services. (s. f.-j). *Clases de almacenamiento de Amazon S3.* https://aws.amazon.com/es/s3/storage-classes/

Amazon Web Services. (s. f.-k). *Tutoriales en video de Amazon S3.* https://aws.amazon.com/es/s3/videos/?nc=sn&loc=6&dn=2

Amazon Web Services. (s. f.-l). *¿Qué es Amazon S3?* https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/

Amazon Web Services. (s. f.-m). *Introducción a Amazon S3*. https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/GetStartedWithS3.html

AWS Events. (2021, febrero 5). *AWS re:Invent 2020: Introducing Amazon Redshift Machine Learning* [Vídeo]. YouTube. https://youtu.be/3BN1w8JUtD4

Chacaltana, G. (2017, mayo 22). *El poder de las bases de datos NoSQL*. https://solocodigoweb.com/blog/2017/05/22/el-poder-de-las-bases-de-datos-nosql/

Chayel, M. (2014, noviembre 26). *ETL Processing Using AWS Data Pipeline and Amazon Elastic MapReduce*. Amazon Web Services. https://aws.amazon.com/es/blogs/big-data/etl-processing-using-aws-data-pipeline-and-amazon-elastic-mapreduce/

Cogent. (2024, septiembre 1). *Data Warehousing in the Cloud: Best Practices*. https://www.cogentinfo.com/resources/data-warehousing-in-the-cloud-best-practices

Datos.gob.es. (s. f.). *Normas técnicas para un correcto gobierno del dato*. https://datos.gob.es/es/documentacion/normas-tecnicas-para-un-correcto-gobierno-del-dato

Dehghani, Z. (2020). *Data Mesh Principles and Logical Architecture*. Martin Fowler. https://martinfowler.com/articles/data-mesh-principles.html

Dehghani, Z. (2022). *Data Mesh: Delivering Data-Driven Value*. O'Reilly Media.

Deloitte. (s. f.). *Big data arquitect*.

Diarioti. (2015, octubre 3). *¿Qué es el almacenamiento distribuido y por qué Google está silenciosa?* https://diarioti.com/brand/storpool/que-es-el-almacenamiento-distribuido-y-por-que-google-esta-silenciosa

DiegoCoronel. (2021, abril 28). *Big Data: Arquitectura e importancia* [Entrada en un foro]. Huawei. https://forum.huawei.com/enterprise/es/Big-Data-Arquitectura-e-importancia/thread/667224227310354432-667212895836057600

Elternativa. (s. f.). *Data Mesh: qué es, ventajas y cómo implementar esta arquitectura de datos*. https://www.elternativa.com/blog-elternativa/data-mesh/

Fernández, Y. (2021, enero 8). *Big Data: qué es y para que sirve*. Xataxa. https://www.xataka.com/basics/big-data-que-sirve

Francis, R., Gupta, R. y Oke, M. (2023). *Amazon Redshift: The Definitive Guide: Jump-Start Analytics Using Cloud Data Warehousing*. O'Reilly Media.

Gil, E. (2016). *Big data, privacidad y protección de datos.* Agencia Española de Protección de Datos. https://www.aepd.es/es/documento/big-data.pdf

Grande, C. y Labella, E. (2022, abril 6). *¿Qué es un Data Mesh y cómo saber si tu empresa lo necesita?* KPMG Tendencias. https://www.tendencias.kpmg.es/2022/04/data-mesh-descentralizacion-organizativa-informacion/

Graph Everywhere. (s. f.). *Bases de datos NoSQL vs SQL | Bases de datos no relacionales vs relacionales*. https://www.grapheverywhere.com/nosql-vs-sql/

Keepler. (2022, enero 26). *Entendiendo el concepto Data Mesh, la nueva tendencia para resolver (casi) todos los problemas de datos*. https://keepler.io/es/2022/01/entendiendo-concepto-data-mesh/

MDCloud. (s. f.). *Dónde almacenar los datos de una empresa.*

IBM. (s. f.). *¿Y si una arquitectura data fabric pudiera guiar la toma de decisiones?* https://www.ibm.com/es-es/data-fabric

Ilimit. (2020/ abril 28). *Bases de Datos SQL vs NOSQL: Diferencias, Ventajas y Mitos.* https://www.ilimit.com/blog/base-de-datos-sql-nosql/

Immon, W. H. (2005). *Building the Data warehouse Fourth Edition*. Willey.

Indra. (s. f.). *¿Cuántas V debería tener el Big Data?* https://www.indracompany.com/es/blogneo/cuantas-v-deberia-big-data

Jineshkumar, P. (2021, agosto 18). *All about AWS S3 Storage*. Jineshkumar Patel. https://blog.jineshkumar.com/all-about-aws-s3-storage

KeepCoing. (s. f.). *¿Qué es la arquitectura Big Data y cómo funciona?* https://keepcoding.io/blog/que-es-arquitectura-big-data/

Kimball, R. (2013). *The Data warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley.

Kumar, P. (2023, marzo 26). *What is data mesh, when should it be used, and what are its challenges?* LinkedIn. https://www.linkedin.com/pulse/what-data-mesh-when-should-used-its-challenges-purushottam-kumar

Laplante, A. (2021). *Data fabric as Modern Data Architecture*. O'Reilly Media.

LisM24. (2021, julio 23). *Big Data – Aldo Pérez_Lisbeth Magallón.* ISSUU. https://issuu.com/lism24/docs/revista_big_data_-_aldop_rez_lisbethmagall_n

López Fandino, V. M. (2023). *Sistemas de big data*. RA-MA.

Lu, H., Foh, C. H., Wen, Y. y Cai, J. (2012). Optimizing content retrieval delay for LT-based distributed cloud storage systems. *2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA*, 1920-1925. DOI 10.1109/GLOCOM.2012.6503396.

Mamani, J. (2023, mayo 30). *Hadoop I: Get in touch with Docker*. Medium. https://medium.com/@jose.rolando.m/hadoop-get-in-touch-with-docker-c55da692f06

Marco de Desarrollo de la Junta de Andalucía. (s. f.). *Conceptos sobre la escalabilidad.* https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/220

Modus. (s. f.). *Data Fabric*. https://www.modus.es/data-fabric/

Moncho Terol. (2022). *Data Mesh: ¿qué es y cómo beneficia tu estrategia de gestión de datos?* ThinkBig. https://blogthinkbig.com/data-mesh-que-es-y-como-beneficia-a-tu-estrategia-de-gestion-de-datos

NetApp. (s. f.). *Almacenamiento de datos unificado*. https://www.netapp.com/es/data-fabric/what-is-data-fabric/

Nextu. (s. f.). *Características del big data.*

Nexus integra. (s. f.). *Big Data vs Inteligencia Artificial*. https://nexusintegra.io/es/big-data-vs-inteligencia-artificial/

Página de Gartner (https://www.gartner.com/en).

Paloma Rojo. (s. f.). *¿Por qué las empresas masivamente interactivas están impulsando la adopción de NoSQL?* DataIQ. https://dataiq.com.ar/blog/por-que-las-empresas-masivamente-interactivas-estan-impulsando-la-adopcion-de-nosql/

Plataforma Agua. (s. f.). *Análisis big data.*

Poccia, D. (2021, mayo 27). *Amazon Redshift ML Is Now Generally Available – Use SQL to Create Machine Learning Models and Make Predictions from Your Data*. Amazon Web Services. https://aws.amazon.com/es/blogs/aws/amazon-redshift-ml-is-now-generally-available-use-sql-to-create-machine-learning-models-and-make-predictions-from-your-data/

PowerData. (s. f.). *Big Data: ¿En qué consiste? Su importancia, desafíos y gobernabilidad*. https://www.powerdata.es/big-data

Pure Storage. (s. f.). *¿Qué es el almacenamiento de datos por niveles?* https://www.purestorage.com/es/knowledge/what-is-tiered-data-storage.html

Reinsel, D., Gantz, J. y Ryding, J. (2018). *The Digitization of the World. From Edge to Core. Seagate*. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

Rigoberto M. (2022, julio 1). *¿Qué es Data Mesh y por qué revoluciona al mundo de los datos?* LinkedIn. https://es.linkedin.com/pulse/qu%C3%A9-es-data-mesh-y-por-revoluciona-al-mundo-de-los-malca-la-rosa

Rios, S. (2020, abril 26). *Los Datos como Servicio y la Modernización de Aplicaciones*. LinkedIn. https://www.linkedin.com/pulse/los-datos-como-servicio-y-la-modernizaci%C3%B3n-de-salvador-rios-sr-/?originalSubdomain=es

SAP. (s. f.). *¿Qué es un data fabric?* https://www.sap.com/spain/insights/what-is-data-fabric.html

SAPTOOLS. (s. f.). *¡Toma la temperatura a tus datos!* https://www.saptools.es/toma-la-temperatura-a-tus-datos/

Stackscale. (2023, junio 22). *3 Tipos de almacenamiento: de archivos, en bloques y de objetos.* https://www.stackscale.com/es/blog/tipos-de-almacenamiento/

Tehreem Naeem. (2024, marzo 21). *Repositorio de datos: definición, tipos y beneficios con mejores prácticas*. https://www.astera.com/es/type/blog/data-repository/

Telefónica Tech. (2021, mayo 9). *La evolución de las bases de datos con propósito IA en entornos Cloud*. https://telefonicatech.com/blog/la-evolucion-de-las-bases-de-datos-con-proposito-ia-en-entornos-cloud

T-Systems. (2018, agosto 20). *Big Data: ventajas del almacenamiento de datos por niveles.*https://www.t-systemsblog.es/big-data-almacenamiento-datos-niveles/

T-Systems. (2020, septiembre 9). *¿Qué es Data Fabric y para qué sirve?* https://www.t-systemsblog.es/que-es-data-fabric-y-para-que-sirve/

Vidal Vidal, L. H. (2019, octubre 7). *Conceptos iniciales sobre la Arquitectura del Big Data y el Gobierno de Datos*. LinkedIn. https://www.linkedin.com/pulse/conceptos-iniciales-sobre-la-arquitectura-del-big-data-vidal-vidal/?originalSubdomain=es

Worlikar, S., Arumugam, T. y Patel, H. (2021). *Amazon Redshift Cookbook: Recipes for building modern data warehousing solution*. Packt Publishing.

Ye, G. (2023, marzo 15). *Data Mesh, la descentralización del dato*. Hiberus blog. https://www.hiberus.com/crecemos-contigo/data-mesh-la-descentralizacion-del-dato/

Tema 4. Computación distribuida. Computación paralela

4.1. Introducción y objetivos

En este tema aprenderás sobre el paradigma del procesamiento de la información que ha sido almacenada en dichos sistemas y cómo supone un cambio en cuanto a la manera de trabajar con respecto a lo que se venía realizando hasta ahora.

Entenderás en qué consiste el procesamiento de Hadoop con MapReduce y verás qué herramientas dentro del ecosistemas de Hadoop ayudan a realizar dicho procesamiento de una manera más sencilla y cómoda siempre usando como base MapReduce. Dentro de estas aplicaciones conocerás algunas, como Hive o Impala, que permiten realizar consultas analíticas de la información almacenada en HDFS a través del procesamiento con MapReduce.

También profundizaremos en el principal de *software* de procesamiento de datos de hoy en día con Apache Spark.

Los **objetivos** que persigue este tema son los siguientes:

- Entender por qué Spark es superior a MapReduce atendiendo a su diseño.
- Identificar los módulos que componen Spark y el propósito de cada uno.
- Conocer la arquitectura y el funcionamiento interno de Spark.
- Practicar con algunas funciones típicas de procesamiento de datos con Spark.
- Conocer la API estructurada de Spark y su principal estructura de datos, el *dataframe*.
- Identificar las ventajas de usar *dataframes* en lugar de RDD (*resilient distributed datasets*).
- Conocer Spark SQL, así como sus similitudes y diferencias con la API estructurada.

- Practicar con algunas funciones típicas de procesamiento de *dataframes*, tanto con la API estructurada como con Spark SQL.
- Introducir al estudiante en los módulos de alto nivel de Spark.
- Presentar las capacidades de Spark MLlib mediante ejemplos de uso.
- Mostrar las ideas básicas y un ejemplo sencillo de Spark Structured Streaming.

4.2. Spark I

En el año 2009, en Berkeley, EE. UU., surgió una nueva propuesta para paliar estas deficiencias, que, poco a poco, ha ido imponiéndose hasta reemplazar completamente a Hadoop (más concretamente, a MapReduce).

Apache Spark es un motor unificado de cálculo en memoria y un conjunto de bibliotecas para procesamiento paralelo y distribuido de datos en clústeres de ordenadores.

Analicemos la **definición:** es un *motor* de cálculo paralelo y distribuido, esto es, un *framework* de propósito general orientado a resolver cualquier tipo de problema y con el que se puede implementar cualquier algoritmo.

- **Unificado:** el motor es único e independiente de cómo utilicemos Spark:
  - Desde la API de *dataframes* (para lenguaje R, Python, Java y Scala).
  - Desde herramientas externas que lanzan consultas SQL contra Spark (Hive, herramientas de *business intelligence* conectadas a Spark como Tableau, PowerBI, Microstrategy, QlikSense, etc.).
  - Desde una instrucción de la API de programación que recibe una consulta SQL, como `string`, tan compleja como sea necesario.
  - Cualquiera de las opciones anteriores se traduce a un grafo de tareas al que Spark aplica optimizaciones de código automáticamente; por tanto, todos (API, aplicaciones BI, SQL, etc.) se benefician de ellas.
- **En memoria**: todos los cálculos se llevan a cabo en memoria y solo se escriben resultados a disco (parciales o finales) cuando el usuario lo indica explícitamente (operación de guardado) o cuando la operación indicada por el usuario requiere

forzosamente realizar movimiento de datos entre nodos (shuffle , el cual se lleva a cabo desde el disco duro del emisor al disco duro del nodo receptor). Además, el movimiento de datos (y las operaciones de acceso a disco que conlleva) solo se produce cuando es irremediable, pero no de manera obligatoria, como ocurría en MapReduce. Esto consigue un rendimiento muy superior en tareas iterativas (varias pasadas sobre los mismos datos, por ejemplo, algoritmos de ML).

Componentes de Spark

Apache Spark consiste en una **API** (bibliotecas de programación) distribuida, mucho más intuitiva que MapReduce. Al igual que este, abstrae todos los detalles de comunicación de red entre las máquinas del clúster, pero, además, opera de manera similar a las consultas SQL tradicionales, como si los datos fuesen tablas (distribuidas). Esto la hace fácil de usar y de aprender.

Spark ofrece distintas API para cuatro **lenguajes:** Java, Scala (la más utilizada en la actualidad por los ingenieros de datos para aplicaciones en producción), Python (paquete de Python PySpark, muy usado por científicos de datos, con una API casi idéntica a la de Scala que, en realidad, no es más que una capa adicional que termina llamando al código de Scala) y R (paquete de R SparkR, con una API muy diferente al resto, más cercana a los comandos típicos de R).

En este tema, usaremos PySpark para todos los ejemplos.

La Figura 1 muestra los principales componentes de Spark.

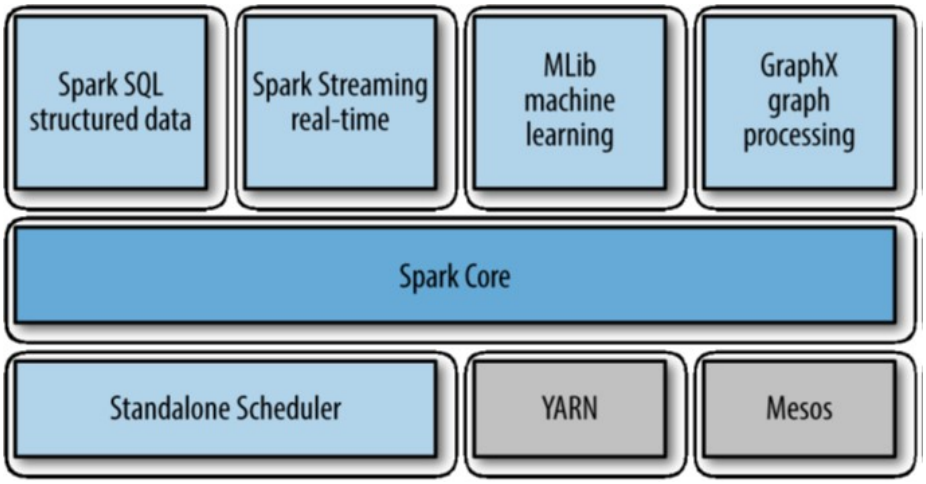


Figura 1. Módulos que componen Apache Spark. Fuente: Chambers y Zaharia 2018.

De estos, el principal es el módulo **Spark Core.** En él se encuentran las estructuras de datos fundamentales de Spark, tales como los RDD, de los que hablaremos más adelante, y las operaciones que se puede llevar a cabo con él. En última instancia, todas las operaciones que se pueden hacer con Spark, independientemente del módulo utilizado o la API donde se encuentren, se traducen automáticamente por parte de Spark a operaciones sobre RDD, que son las únicas que sabe ejecutar el motor.

Los tres módulos inferiores representan tres **gestores de recursos** de un clúster sobre los que puede ejecutarse Spark. Un gestor de recursos se encarga de asignar máquinas, CPU y memoria principal a Spark, de forma que disponga de un determinado número de nodos y de que, en cada uno, existan suficientes recursos para ejecutar la aplicación que hemos implementado con Spark.

El resto de los módulos cumplen las siguientes **funciones:**

- **Spark SQL y API estructurada** es una API con una serie de funciones para manejar tablas de datos distribuidas, estructuradas en columnas con nombre y tipo, denominadas *dataframes*. Un *dataframe* proporciona un nivel de abstracción adicional sobre un RDD, al cual recubre. Este módulo incluye también una función para ejecutar sentencias SQL sobre las mismas estructuras de datos distribuidas.

- **Spark Streaming** es el módulo para operar de manera distribuida sobre datos en tiempo real, según los vamos recibiendo (*stream* de datos). A partir de Spark 2.0, este módulo ha sido reemplazado por **Spark Structured Streaming,** que simplifica el procesamiento de flujos de datos en tiempo real.

- **SparkML** contiene implementaciones distribuidas de algoritmos de ML.

- **Spark GraphX** es el módulo de procesamiento de grafos, que se representan mediante RDD. Contiene algunos algoritmos de camino mínimo y similares. Actualmente, este módulo ha quedado obsoleto y ha sido reemplazado *de facto* por el paquete GraphFrames, que representa un grafo como una pareja de *dataframes*, con los nodos y los arcos respectivamente. Empezó como un proyecto separado de Spark y, finalmente, ha sido integrado en las versiones más recientes.

Arquitectura de Spark

La Figura 2 muestra la arquitectura de una aplicación que utiliza Spark al ejecutarse en un clúster. No debemos olvidar que, al escribir una aplicación en Spark, en realidad, estamos escribiendo una aplicación **secuencial** (no paralela) utilizando la biblioteca de Spark para el lenguaje en el que estemos programando (Java, Scala, Python o R). Cuando ejecutamos el código de este programa (proceso que se denomina *driver*), lo hacemos sobre una máquina concreta, que puede ser interna o externa al clúster. Por ejemplo, podríamos escribir un programa en Spark y ejecutarlo en nuestro ordenador portátil, y, desde aquí, el programa driver podría conectarse a un clúster de Spark remoto.

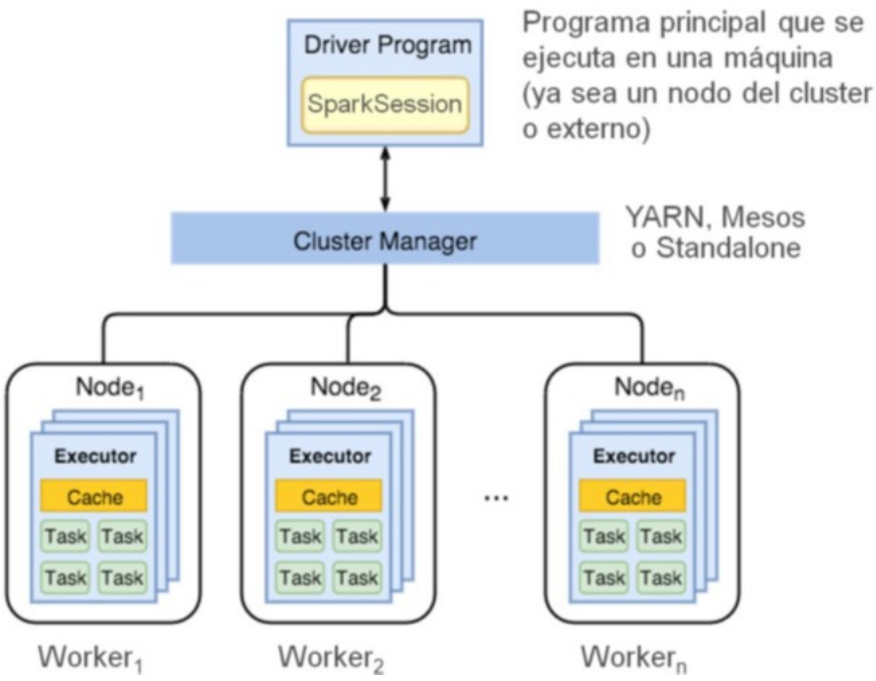


Figura 2. Arquitectura de una aplicación en Spark al ejecutarse en un clúster. Fuente: Datastrophic, 2016.

En el transcurso de la **ejecución,** normalmente necesitaremos crear un objeto denominado `sparkSession`, de la clase `SparkSession` de Spark. En el momento de crear este objeto, hay que indicar dónde (en qué dirección IP y en qué puerto) existe un clúster de Spark configurado. Ciertas aplicaciones, como, por ejemplo, Jupyter Notebook (cuando está configurado para Spark) o la línea de comandos (intérprete) de Spark, tanto en Scala (`spark-shell`) como en Python (`pyspark-shell`), ya nos dan un objeto `sparkSession` creado al arrancar la aplicación. No obstante, al iniciar el *shell*, debemos pasarle la configuración relativa al clúster para que la `sparkSession` se cree correctamente.

De lo contrario, se estará ejecutando en local, sin **conexión con un clúster.** De esta manera, la `sparkSession` establece comunicación con el gestor de clúster para poder enviar tareas a los *workers* (nodos del clúster disponibles para Spark). A partir de este momento, podemos utilizar dichos recursos para las sentencias que requieran ejecución distribuida.

Es importante recalcar esto último: la **ejecución** de un programa en Spark es **secuencial,** en una sola máquina, tal como sería cualquier otro programa en el lenguaje elegido, excepto cuando el flujo de programa llega a ciertas funciones específicas de Spark que desencadenan **ejecución distribuida,** que son la mayoría (pero no todas) de las que forman parte de la API de Spark. Muchas se aplican sobre el objeto `sparkSession` y otras sobre estructuras de datos distribuidas que hayamos ido creando en el programa, como, por ejemplo, RDD o *dataframes* de Spark.

En el momento de crear el objeto `sparkSession`, hemos indicado una configuración con el número de nodos, la memoria RAM y el número de núcleos físicos que necesitamos reservar en cada nodo. En un nodo, estos recursos constituyen lo que se denomina un ***executor*:** un proceso de la JVM (máquina virtual de Java) que se ejecuta en el nodo y que ocupa los recursos indicados (núcleos, RAM, disco duro).

El **proceso ejecutor** es creado por el gestor de clúster cuando arranca nuestra aplicación de Spark y muere cuando la aplicación finaliza (ya sea con éxito o por alguna condición imprevista que provoca que toda la aplicación termine abruptamente). Cada ejecutor queda preparado para ejecutar tareas de Spark, que es la unidad mínima de ejecución de trabajos. Cada tarea requiere un núcleo libre para ejecutarse, por lo que, si un *executor* tiene reservados cuatro núcleos, podrá ejecutar cuatro tareas en paralelo. Detallaremos esto más adelante. Cada uno de los **nodos del clúster** en los que se crean ejecutores se denomina *worker*.

RDD

Los RDD constituyen la abstracción fundamental de Spark.

Un RDD es una colección no ordenada (*bag*) de objetos que está distribuida en la memoria RAM de los nodos del clúster.

La colección está dividida en particiones, cada una de las cuales está en la memoria RAM de un nodo distinto del clúster. Al desgranar el nombre, tenemos que:

- ***Resilient* ('resistente' o 'adaptable').** Es posible reconstruir un RDD que estaba en memoria, a pesar de que una de las máquinas falle, gracias al DAG de ejecución (*directed acyclic graph* o grafo de ejecución), que veremos más adelante.

- ***Distributed* ('distribuido').** Los objetos de la colección están divididos en particiones que están distribuidas en la memoria principal de los nodos del clúster. La colección no está ordenada, por lo que no se puede acceder mediante una posición a objetos individuales.

- ***Dataset*.** La colección representa un conjunto de datos que estamos procesando de forma paralela y distribuida, para transformarlos, calcular agregaciones, etc.

La Figura 3 representa tres RDD diferentes, distribuidos en la memoria RAM de un clúster de cuatro nodos. No todos los RDD presentan el mismo número de particiones. En la Figura 3, uno de los RDD tiene solo dos, otro tiene tres y, otro, cuatro. La idea es similar al almacenamiento de los ficheros en HDFS, donde están distribuidos entre los discos duros de los nodos, con la diferencia de que, en este caso, los RDD están distribuidos en la memoria RAM de los nodos y, además, **no hay replicación de cada partición.**

Si un nodo falla, es posible reconstruir las particiones que estuvieran en ese momento en su memoria principal gracias al **DAG,** que mantiene la traza de cómo se construyeron. El DAG es otro mecanismo que proporciona robustez sin la necesidad de replicar las particiones de un RDD. Es preciso indicar también que, a pesar de que la Figura 3 muestra una sola partición de cada RDD en cada nodo, lo habitual es que, en la memoria de un mismo nodo, haya numerosas particiones de un mismo RDD (de hecho, es normal que los RDD que se van calculando tengan decenas o cientos de particiones).

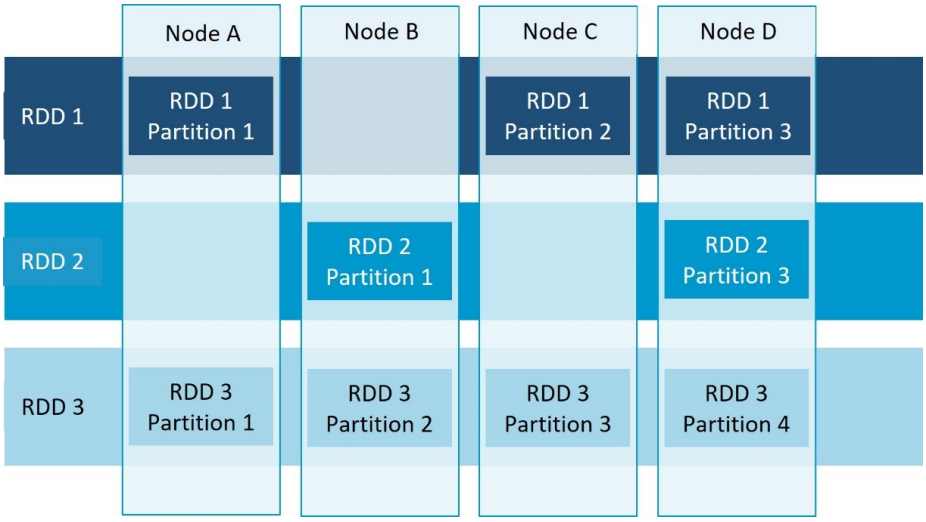


Figura 3. Representación de tres RDD en un clúster de Spark con cuatro *workers*. Fuente: elaboración propia.

En relación con los RDD, cabe destacar dos conceptos:

- **Inmutabilidad.** El contenido de un RDD no puede modificarse una vez creado. Lo que hacemos es aplicar transformaciones a los RDD para obtener otros nuevos, pero los datos del RDD existente no se alteran. La idea de la ejecución es que, cuando aplicamos una transformación, se ejecuta en paralelo sobre todas las particiones del RDD, de manera transparente al programador, para dar lugar a un nuevo RDD, cuyas particiones son el resultado de aplicar la transformación sobre cada una de las particiones del original.

> Ejemplo
>
> Dado un RDD de números reales, para multiplicar cada elemento por dos, aplicamos una transformación que actúa en cada elemento y lo multiplica por dos. Spark lleva nuestro código de la transformación (lo serializa y lo envía por la red) a cada uno de los nodos del clúster donde haya particiones de ese RDD y lo ejecuta en ellos para que actúe en cada elemento de esa partición. Todo de manera transparente al programador.

Una vez más, los datos son el centro, lo más importante; no se mueven salvo que sea imprescindible. Ciertos tipos de transformaciones no requieren movimiento de datos, pero otros sí, como veremos después.

- **Partición.** Es un subconjunto de los objetos de un RDD que están presentes en un mismo nodo. Es la unidad de datos mínima sobre la que se ejecuta una tarea de transformación de manera independiente al resto de particiones. Idealmente, tendría que haber, al menos, tantas particiones como núcleos físicos (procesadores) disponibles en nuestro clúster. De esta manera, nos aseguramos de que todos los núcleos estarán ocupados, incluso en el caso de que nada más que nuestra aplicación estuviese empleando ese clúster. Lo habitual es que haya muchas particiones de un mismo RDD en cada nodo, en un número muy superior al de procesadores existentes en el ellos. Se recomienda que cada RDD esté dividido en un número de particiones que sea entre el doble y el triple que el número de procesadores del clúster.

Originalmente, los programadores de Spark trabajaban con RDD. Sin embargo, en Spark 1.6, se introdujeron los ***dataframes*****,** que definiremos más adelante. Desde Spark 2.0, los propios creadores recomiendan encarecidamente no utilizar los RDD, sino siempre *dataframes* y su API correspondiente. Aparte de la facilidad de uso, el motivo fundamental es que los *dataframes* están sujetos a importantes optimizaciones automáticas de código por parte del analizador de Spark, llamado Catalyst. Los RDD no están sujetos a estas optimizaciones y la ejecución es sensiblemente más lenta. Todos los ejemplos los presentaremos con PySpark.

Operaciones de transformación y de acción

Podemos realizar dos **tipos de operaciones** cuando usamos la API de Spark:

- **Transformación.** Es una operación que se ejecuta sobre un RDD y devuelve un nuevo RDD, en el que sus elementos se han modificado de algún modo. Son *lazy* (perezosas), es decir, no se ejecuta nada hasta que Spark encuentra una acción. Mientras tanto, Spark simplemente añade la transformación al grafo de ejecución (el DAG), que mantiene la trazabilidad y permite la *resiliency*. El DAG guarda toda la secuencia de transformaciones que se realizaron para obtener cada RDD concreto que se vaya creando en nuestro código.

  Si la transformación no implica `shuffle` (movimiento de datos entre nodos), se denomina `narrow` y cada partición da lugar a otra en el mismo nodo. Hay que recordar que una operación `shuffle` implica una escritura previa de los datos en el disco duro local del nodo emisor y después en el disco local del nodo receptor, de manera transparente al programador.

- **Acción.** Recibe un RDD y calcula un resultado (generalmente, un tipo simple, enteros, *doubles,* etc.) y lo devuelve al *driver* (programa principal, que corre en una máquina). El tipo de dato devuelto al *driver* no es un RDD, sino un tipo nativo del lenguaje que estemos utilizando (Java/Scala/Python/R).

Importante: el resultado de la acción debe caber en la memoria de la máquina donde se está ejecutando el proceso *driver.*

Una **acción** desencadena instantáneamente el cálculo de toda la secuencia de transformaciones intermedias y la materialización de los RDD involucrados.

Una vez materializado un RDD, se aplica la **transformación** que toque, según indica el DAG, para generar el siguiente RDD y el anterior se libera (no permanece en la memoria RAM, salvo que se indique expresamente mediante el método `cache()`). Un

RDD cacheado permanece materializado en la RAM de los nodos y no es necesario recalcularlo después de que se haya materializado la primera vez.

Por defecto, el **punto de partida** del DAG son operaciones de lectura de datos, bien desde una fuente de datos, como, por ejemplo, HDFS o el sistema Amazon S3, bien desde alguna base de datos (distribuida o no) o similar. Si ningún RDD intermedio ha sido cacheado, cualquier operación que haga referencia a un RDD exigirá reconstruir toda la secuencia de transformaciones previas a dicho RDD, empezando por la lectura de los datos, excepto que alguno de los RDD intermedios haya sido cacheado. Esto hace que la secuencia empiece en el RDD cacheado y no haya que remontarse al origen de datos.

La **utilización del DAG** sirve para poder reconstruir cualquier RDD de una secuencia de transformaciones, tanto si la necesidad de materializar el RDD se debe a que hacemos referencia a él varias veces a lo largo de nuestro código como si obedece a que, durante el procesamiento, falla algún nodo y el contenido de su memoria RAM se pierde. Gracias al DAG es posible reconstruir cualquier partición concreta de cualquier RDD y volver a lanzar la secuencia de transformaciones en otros nodos que sí estén activos.

Existen ciertas operaciones de la API de Spark que **no son transformaciones ni acciones,** como, por ejemplo, `cache()`, sino que sirven para configurar, habilitar o deshabilitar ciertos comportamientos, o para obtener características relativas a la distribución física de un RDD (consultar el número de particiones que tiene, consultar si está cacheado, consultar el esquema —nombres y tipos de las columnas— de un *dataframe*, etc.

Transformaciones más habituales con RDD

Recordemos que, para todas las operaciones que reciben una función, Spark serializa el código de la función y la envía por red a los nodos, donde es ejecutada.