Cloud Computing: Servicios y Aplicaciones

Procesamiento y minería de datos en Big Data con Spark sobre plataformas cloud

Tabla de contenido:

Introducción

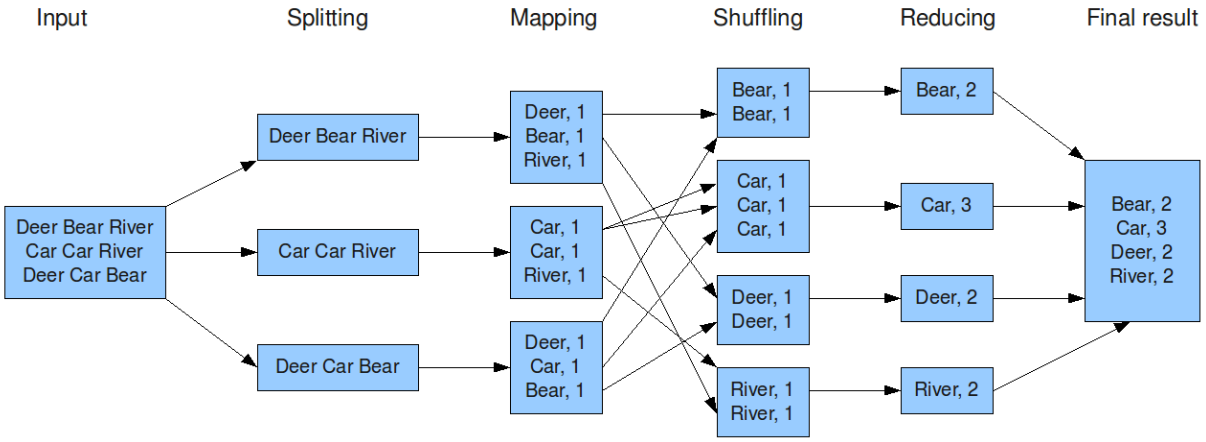
Nos encontramos dentro de un mundo hiper-conectado, donde miles de millones de dispositivos de tipos muy diversos están generando ingentes cantidades de datos cada segundo: desde aplicaciones móviles a dispositivos IoT, pasando por servicios en la nube, aplicaciones corporativas, o el mercado de valores entre muchas otras. Igualmente son cientos de miles de Terabytes los producidos por aplicaciones, sistemas y desarrollos científicos relacionados con biotecnología, física de partículas, astrofísica, etc., de modo que tener los datos disponibles y poder procesarlos de una forma cómoda y eficiente es vital para obtener resultados, y así, tener la capacidad **de tomar decisiones** o **predecir valores futuros** de una magnitud. El problema principal que existe cuando se trabaja con volúmenes de datos muy grandes es que el procesamiento en ordenadores convencionales no es posible. Basándonos en la infraestructura disponible, es necesario encontrar técnicas y modelos de computación que permitan captar, almacenar y procesar estos conjuntos de datos masivos. Además, el crecimiento continuado del ritmo de generación de datos acentúa la necesidad de un enfoque completamente escalable. En este sentido, la única plataforma capaz de dar soporte a estas necesidades es la que ofrece el paradigma Cloud Computing. Sin embargo, para completar el flujo se hacen necesarias nuevas formas de diseñar programas adaptados a los nuevos requisitos. La respuesta más efectiva a este desafío ha sido el modelo de programación MapReduce, y extensiones de éste. Desde hace ya más de 10 años, diferentes *frameworks* y plataformas han ido apareciendo implementado este modelo de programación. A continuación describimos algunos de los paradigmas, plataformas y frameworks más utilizados en Big Data.

MapReduce

MapReduce es una técnica de procesamiento y un modelo de programación para computación distribuida. La idea principal de MapReduce se basa en estructurar cada programa sobre dos funciones importantes: Aplicación (Map) y Reducción (Reduce). Map toma un conjunto de datos y lo convierte en otro conjunto de datos, donde los elementos individuales se descomponen en pares clave/valor. En segundo lugar, la tarea de reducir, que toma la salida de cada map como entrada y combina esos pares de datos agregándolos en un conjunto más pequeño de pares. Tal y como expresa el nombre MapReduce, la tarea de reducir siempre se realiza después del trabajo del Map.

La principal ventaja de MapReduce es que es fácil de escalar el procesamiento de datos en múltiples nodos de computación. Bajo el modelo de MapReduce, los primitivos procesadores de datos se llaman Mappers y Reducers. Descomponer una aplicación de procesamiento de datos en Mappers y Reducers a veces no es trivial. Pero, una vez que estructuramos una aplicación en el formato MapReduce, escalar la aplicación para que funcione sobre cientos, miles o incluso decenas de miles de máquinas en un clúster es simplemente un cambio de configuración. Esta escalabilidad tan sencilla es lo que ha atraído a muchos programadores a usar el modelo de MapReduce cuando se trata de volúmenes de datos enormes.

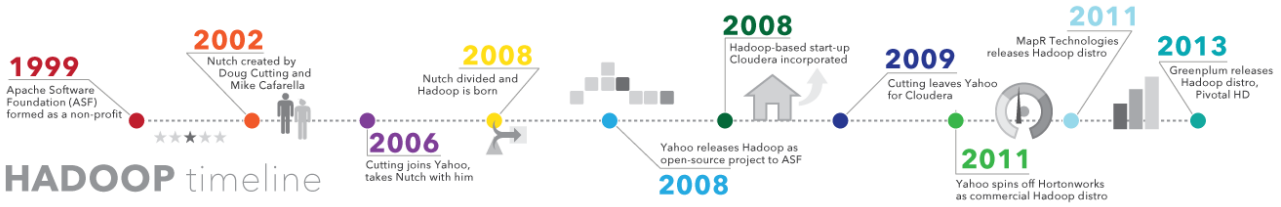
MapReduce es adecuado problemas que se pueden dividir en unidades independientes, pero no es eficiente para realizar tareas iterativas e interactivas. MapReduce trabaja con muchos archivos. Como los nodos no se intercomunican salvo a través de procesos de clasificación y mezcla, los algoritmos iterativos requieren múltiples fases de mapeo-mezcla/clasificación-reducción para completarse. Esto da origen a múltiples archivos entre fases de MapReduce y no es eficiente para el cómputo avanzado.



Hadoop y Spark son dos de las plataformas y frameworks que implementan el paradigma de procesamiento MapReduce.

Hadoop[1]

Hadoop es una estructura de software de código abierto para almacenar datos y ejecutar aplicaciones en clústeres de equipos convencionales. Proporciona almacenamiento masivo para cualquier tipo de datos, enorme poder de procesamiento y la capacidad de procesar tareas o trabajos concurrentes virtualmente ilimitados.



Uno de estos proyectos fue un buscador Web de código abierto llamado Nutch – idea original de Doug Cutting y Mike Cafarella. Deseaban generar resultados de búsquedas en la Web a mayor velocidad distribuyendo datos y cálculos en diferentes computadoras de modo que se pudieran procesar múltiples tareas de manera simultánea. Durante este tiempo, estaba en progreso otro proyecto de buscador llamado Google. Éste se basaba en el mismo concepto – almacenar y procesar datos de manera distribuida y automatizada de modo que se pudieran

[1] https://www.sas.com/es_es/insights/big-data/hadoop.html#hadooptechnical

generar resultados de búsquedas en la Web a mayor velocidad. En 2008, Yahoo presentó Hadoop como proyecto de código abierto. Posteriormente, su evolución y desarrollo fueron acogidos por la Apache Software Foundation, como uno de sus proyectos más exitosos.

- Capacidad de almacenar y procesar enormes cantidades de cualquier tipo de datos, al instante. Con el incremento constante de los volúmenes y variedades de datos, en especial provenientes de medios sociales y la Internet de las Cosas (IoT), ésa es una consideración importante.
- Capacidad de cómputo. El modelo de cómputo distribuido de Hadoop procesa Big Data a gran velocidad. Cuantos más nodos de cómputo utiliza usted, mayor poder de procesamiento tiene.
- Tolerancia a fallos. El procesamiento de datos y aplicaciones está protegido contra fallos del hardware. Si falla un nodo, los trabajos son redirigidos automáticamente a otros modos para asegurarse de que no falle el procesamiento distribuido. Se almacenan múltiples copias de todos los datos de manera automática.
- Flexibilidad. A diferencia de las bases de datos relacionales, no tiene que procesar previamente los datos antes de almacenarlos. Puede almacenar tantos datos como desee y decidir cómo utilizarlos más tarde. Eso incluye datos no estructurados como texto, imágenes y videos.
- Bajo costo. La estructura de código abierto es gratuita y emplea hardware comercial para almacenar grandes cantidades de datos.
- Escalabilidad. Puede hacer crecer fácilmente su sistema para que procese más datos con sólo agregar nodos. Se requiere poca administración.

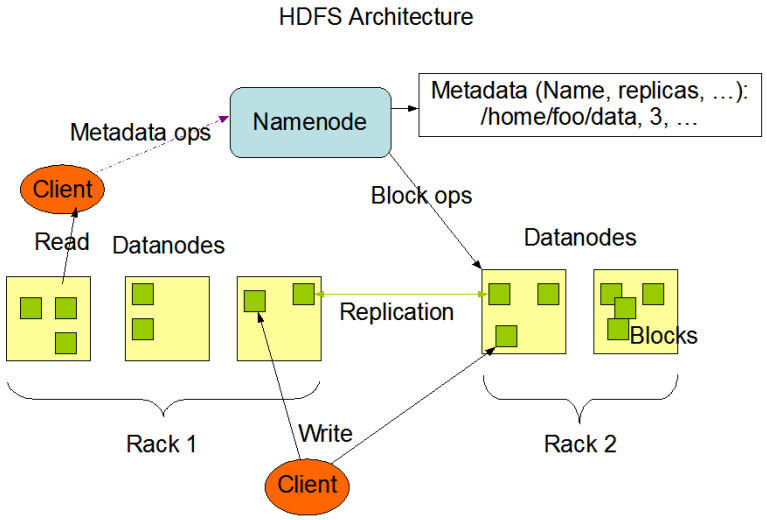
Un elemento clave para los sistemas de procesamiento de datos a gran escala es que necesitan de un soporte de almacenamiento acorde con las características tanto del paradigma como de la plataforma distribuida, de modo que uno de los componentes básicos de este tipo de tecnologías es el sistema de ficheros distribuido HDFS.

HDFS

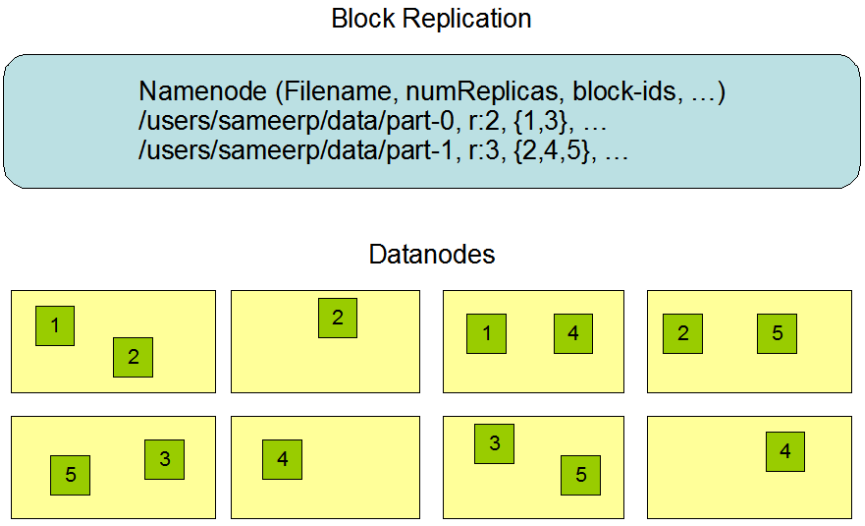
El Sistema de Archivo Distribuido Hadoop (HDFS) es un sistema de archivos distribuido diseñado para funcionar con hardware convencional. Tiene muchas similitudes con los sistemas de archivo distribuidos existentes, pero también hay diferencias significativas. HDFS es altamente tolerante a fallos y está diseñado para ser desplegado en hardware de bajo costo. HDFS proporciona un acceso de alto rendimiento a los datos de las aplicaciones y es adecuado para aplicaciones que procesan grandes conjuntos de datos. Este sistema flexibiliza la gestión de ficheros y grandes volúmenes de datos para permitir el acceso en flujo a los datos del sistema de archivos. HDFS se construyó originalmente como infraestructura para el proyecto de motor de búsqueda web de Apache Nutch, comentado anteriormente como Hadoop. Actualmente HDFS es ahora un subproyecto de Apache Hadoop.

HDFS tiene una arquitectura maestro/esclavo. Un grupo de HDFS consiste en un único NameNode, un servidor maestro que administra el espacio de nombres del sistema de archivos y

regula el acceso a los archivos por parte de los clientes. También existen varios DataNodes, generalmente uno por cada nodo del clúster, que administran el almacenamiento de los respectivos nodos en los que se ejecutan. HDFS expone un espacio de nombres del sistema de archivos y permite que los datos del usuario se almacenen en archivos. Internamente, un archivo se divide **en uno o más bloques y estos bloques se almacenan en un conjunto de DataNodes**. El NameNode ejecuta las operaciones del espacio de nombres del sistema de archivo como abrir, cerrar y renombrar archivos y directorios. También determina la asignación de los bloques a los DataNodes. Los DataNodes son responsables de atender las peticiones de lectura y escritura de los clientes del sistema de archivos. También realizan la creación, borrado y replicación de bloques según las instrucciones del NameNode. En la imagen inferior se puede observar la arquitectura de HDFS:

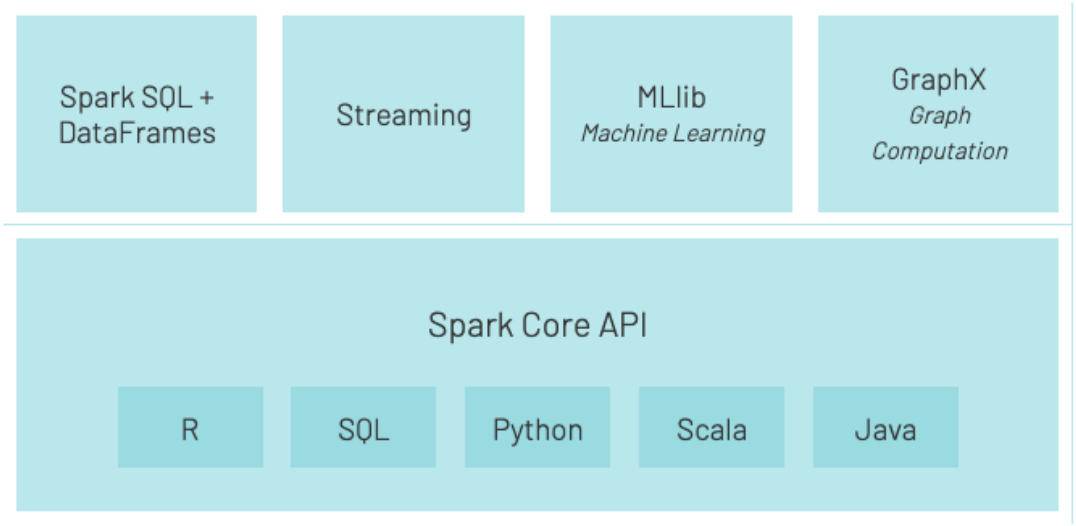


HDFS está diseñado para almacenar los datos de forma fiable a lo largo de un cluster de cualquier tamaño y características. En la imagen inferior se puede ver un ejemplo de la replicación de los bloques de datos:



Spark

Desde su lanzamiento, Apache Spark, ha sido adoptado rápidamente por empresas de una amplia gama de industrias y prácticamente se ha convertido en el estándar de-facto para el procesamiento de datos de gran volumen. Empresas de Internet tan conocidas como Netflix, Yahoo y eBay han desplegado Spark a escala masiva, procesando colectivamente múltiples petabytes de datos en clusters de más de 8.000 nodos. Se ha convertido rápidamente en la mayor comunidad de código abierto en Big Data, con más de 1000 colaboradores de más de 250 organizaciones. Apache Spark está formado por el siguiente ecosistema:



Entre las características de Spak cabe mencionar:

- Velocidad: Diseñado de abajo hacia arriba para el rendimiento, Spark puede llegar a ser 100 veces más rápido que Hadoop para el procesamiento de datos a gran escala explotando en la computación de la memoria y otras optimizaciones. Spark también es rápido cuando los datos se almacenan en el disco, y actualmente tiene el récord mundial de ordenación a gran escala en el disco.
- Facilidad de uso: Spark tiene APIs fáciles de usar para operar en grandes conjuntos de datos. Esto incluye una colección de más de 100 operadores para transformar datos y API de marcos de datos familiares para manipular datos semiestructurados. Un ejemplo sencillo en Python de la expresividad de su API puede verse en este código, que permite consultar datos de una forma muy flexible (lee un json, selecciona los datos con age>21 y luego devuelve la columna (path en json / key) name.first):

```
df = spark.read.json("logs.json")
df.where("age > 21").select("name.first").show()
```

- Un motor unificado: Spark viene empaquetado con bibliotecasd de nivel superior, incluyendo soporte para consultas SQL, transmisión de datos, aprendizaje automático y procesamiento de gráficos. Estas bibliotecas estándar aumentan la productividad de los

desarrolladores y pueden combinarse perfectamente para crear flujos de trabajo complejos.
- Funciona en cualquier plataforma: Tiene soporte para HDFS Hadoop, Apache Mesos, Kubernetes, Cassandra, Hbase, etc.

Dentro del ecosistema cabe destacar MLLib, que contiene muchos algoritmos y utilidades para Big Data:

- Clasificación: regresión logística, Bayes ingenuo, ...
- Regresión: regresión lineal generalizada, regresión de supervivencia, ...
- Árboles de decisión, bosques aleatorios y árboles con gradientes
- Recomendación: alternar los mínimos cuadrados (ALS)
- Agrupación: Medios K, mezclas gaussianas (GMM), ...
- Modelización de temas: asignación de Dirichlets latentes (LDA)
- Conjuntos de elementos frecuentes, reglas de asociación y minería de patrones secuenciales

Además, de modo similar al que se trabajó con AirFlow, Spark tiene utilidades de flujo de trabajo de ML incluyen:

- Transformaciones de características: estandarización, normalización, hashing, ...
- Construcción de pipes en ML
- Evaluación de modelos y ajuste de hiper parámetros
- Persistencia del ML.

Objetivos de la práctica

Los objetivos formativos de esta práctica se centran en que el alumno adquiera los siguientes conocimientos y habilidades:

- Conocimiento de la infraestructura de cómputo para Big Data y su despliegue en plataformas de cloud computing.
- Manejo del sistema HDFS.
- Conocimiento de frameworks para el procesamiento de los datos de gran volumen y aplicación de técnicas de Minería de Datos para la extracción de conocimiento.
- Uso de las herramientas básicas proporcionadas por la MLLib de Spark.

La distribución prevista de contenidos a lo largo de las sesiones es la siguiente:

- HDFS [Primera sesión] https://github.com/manuparra/TallerH2S
  - Manejo y gestión de ficheros en HDFS←→ Local
  - Operaciones básicas para el trabajo con ficheros en HDFS.

- Plataforma SPARK [Primera y segunda sesión]
  https://github.com/manuparra/taller_SparkR
  https://github.com/manuparra/TallerH2S
  https://github.com/manuparra/taller-bigdata-con-r
  - Spark Shell (para Python, Scala y R). Cómo crear el entorno de trabajo para cada uno de los lenguajes e intérpretes. Conexión al servicio del Cluster con YARN.
  - Los DataFrames y el uso/gestión de SparkDataFrames. Componentes del framework que proporciona Spark para el trabajo con datos: Biblioteca de componentes.
  - Consultas a datos usando SparkSQL.
  - Implementación de flujos de trabajo Spark para Minería de Datos con MLLib
- Minería de datos y Machine Learning con Spark desde servicios de Cloud Computing públicos:
  - AZURE: https://github.com/manuparra/starting-bigdata-aws
  - Como desplegar un cluster en Kubernetes. https://github.com/manuparra/Spark+Kubernetes

Descripción de la práctica

En el desarrollo de esta práctica se estudiarán y pondrán en uso diferentes métodos y técnicas de procesamiento de datos para grandes volúmenes de datos. En concreto, el objetivo final será la resolución de un problema de clasificación a través del desarrollo de distintos modelos. Este trabajo implica el diseño conceptual, la programación y el despliegue de distintos modelos, apoyándose en los métodos implementados en la biblioteca MLLib. Para ello habrá que usar hábilmente distintas herramientas de los ecosistemas de Hadoop y Spark, desplegadas sobre distintos escenarios. Una vez diseñados e implementados se comparará el rendimiento de los distintos clasificadores para identificar cuál resulta ser el más adecuado para el problema en cuestión realizando un estudio empírico comparativo.

Para el desarrollo de la práctica se usará el cluster hadoop.ugr.es, que provee de HDFS y Spark ya preparado y listo para trabajar sobre él, y que proporciona todas las herramientas para el procesamiento. También se probará sobre otras plataformas: Azure, DataBricks, ...

El problema a considerar es la realización de distintas tareas de análisis de datos sobre un conjunto de datos concreto. Más concretamente, el conjunto define un problema de clasificación y se trata de construir varios clasificadores que los resuelvan. Está disponible en el siguiente directorio de HDFS:

```
Cabecera: hdfs://user/datasets/ecbdl14/ECBDL14_IR2.header
Datos: hdfs://user/datasets/ecbdl14/ECBDL14_IR2.data

    600 Atributos, 1 de clase
```

```
2060000 registros
4Gbytes
```

```
Nota: Puedes preprocesarlo y hacer primero una reducción para
poder manejarlo y extraer las columnas.
```

Para la realización de las tareas es necesario tener en cuenta las siguientes indicaciones:

- Se usará como lenguajes de programación Scala, Python, Java o R.
- Se procesa el fichero de datos, para crear uno nuevo filtrado que conste de:
  - Las 6 primeras columnas y la variable de clase (class) (total 6 +1 columnas). ***Cada alumno recibirá una serie de columnas específicas*** e individuales, por lo que los resultados de la clasificación serán diferentes para cada alumno. Este listado se encuentra en Prado.
  - El nuevo dataset se llamará filteredC.small.training → copia a HDFS usuario.
- ***El problema de clasificación no está equilibrado. Es necesario preprocesarlo para paliar este inconveniente con alguna técnica como ROS o RUS (random oversampling or random undersampling).***
- Aplicar al menos 3 técnicas de construcción de clasificadores (Naive Bayes, RF, LR, SVM, ...) de la MLLib de Spark. La lista de todos los algoritmos está disponible en: https://spark.apache.org/docs/latest/ml-classification-regression.html.
- Es necesario probar al menos 2 parametrizaciones de cada uno de los algoritmos.
- Para evaluar los clasificadores construidos se realizará un breve estudio experimental. Se trata de construir clasificadores (sobre partición de entrenamiento) y evaluarlos (sobre partición de prueba). Los resultados se recogerán en una tabla que incluirá todos los modelos y combinaciones de hiper parámetros consideradas.
- Finalmente, analizar los resultados obtenidos. Identificar qué algoritmo y parametrización obtiene mejores resultados.

Entrega y evaluación de la práctica

El trabajo realizado en esta práctica es evaluable y formará la segunda parte de la calificación de la asignatura en Prácticas. Los detalles de la entrega y evaluación se incluyen a continuación.

Entrega

Para la entrega de esta práctica:

1. **Se subirá la documentación a PRADO en la actividad "Práctica 4 Spark" en un único fichero en formato PDF, con el nombre "Práctica 4 Spark.pdf".**
   a. Esta documentación debe incluir las siguientes partes:
      i. Portada, con autor, DNI (ID), curso, asignatura.
      ii. Introducción / Descripción de la práctica.

iii. Resolución del problema de clasificación (sin código completo, solo los extractos más representativos):
    1. Explica cada una las tareas, cómo la has resuelto, además del flujo de procesamiento que usas:
        a. Detalla los pasos que has tenido en cuenta para cada tarea de procesado y métodos utilizados.
        b. Los resultados de los clasificadores para cada parametrización.
        c. Código fuente desarrollado. Ya sea debidamente organizado y empaquetado en un fichero .zip, o en vuestra cuenta en betatun.ugr.es.
    2. Conclusiones.

2. Criterios de evaluación:
    a. Se valorará la completitud y calidad de la documentación que se aporta con la práctica.
    b. Se valorará la capacidad de diseñar un flujo de trabajo automatizado con Spark.
    c. Se valorarán las mejoras introducidas en el código para incrementar el rendimiento de la clasificación.
    d. Se valorarán las propuestas sobre el el preprocesamiento de los datos antes de la aplicación de los métodos de ML.
    e. Se valorará el uso de Scala (programación funcional) y Python.

Fecha

Fecha límite de entrega de la documentación y código fuente: **15 Junio de 2021 a las 23:55.**