Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [ ]:

```
NAME = "Vicent Ripoll Ramírez"
COLLABORATORS = ""
```

PEC 2: Ejecución de trabajos mediante el gestor de recursos YARN

0 Nota

En muchos de los ejercicios se deberán realizar **capturas de pantalla que justifiquen las respuestas**. Las capturas de pantalla se pueden realizar con las herramientas del sistema operativo que estéis utilizando. Para copiar la imagen en el portapapeles podéis utilizar diferentes métodos: "Herramienta de Recortes" en Windows, "Imprimir pantalla", Ctrl+C al seleccionar una imagen, etc.... Las imágenes una vez capturadas se pueden pegar directamente en las celdas de respuesta, mediante Ctrl+V o con el menú contextual que aparece con el botón derecho del ratón, lo que pegará la imagen que esté en el portapapeles. Para ver la imagen se debe ejecutar la celda.

1 Introducción

Hasta ahora en los ejercicios realizados hemos visto como ejecutar trabajos en un entorno Big Data pero no nos hemos fijado en como estos trabajan, interactúan y se ejecutan dentro del sistema. Este es precisamente el objetivo de este ejercicio.

Antes de empezar con el anunciado de la PEC, vamos a revisar un poco la composición del sistema Big Data que estáis usando. En la siguiente imagen, mediante el Cloudera Manager, el gestor del sistema que utilizamos[CDH Components] (https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html) , podemos ver como está configurado el gestor de recursos Yarn. Es interesante ver (Figura 1) que en nuestro caso el clúster está compuesto por 4 máquinas, una principal, y más grande, y tres más modestas. Podéis ver las características de las máquinas en las diversas columnas de la izquierda. En vuestro caso, cuando accedéis al sistema, lo hacéis mediante la máquina “eimtcld.uoc.edu”.

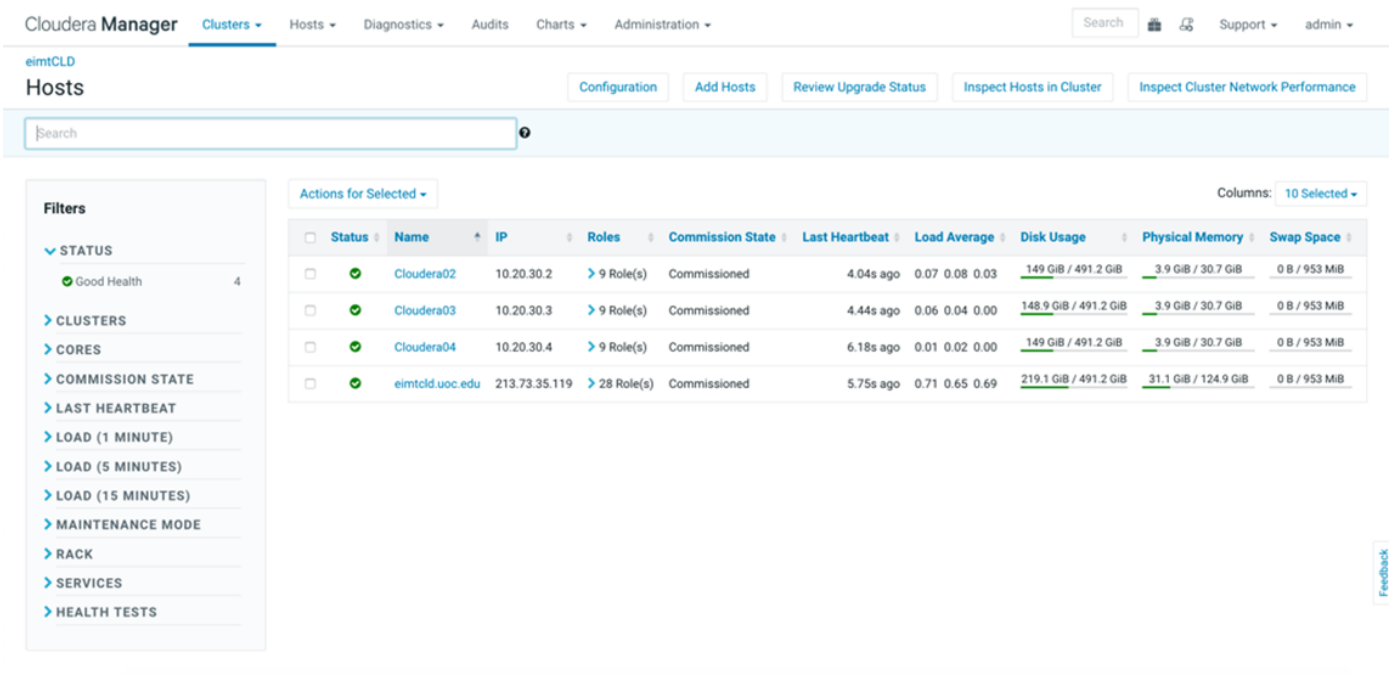


Figura 1: Descripción del sistema Cloudera instalado para la realización del las prácticas.

En la Figura 2 podéis ver la configuración del resource manager usado, el Yarn. Vemos que cada una de las máquinas está corriendo un “Node manager” de Yarn gestionado por el “Resource Manager” que se ejecuta en la máquina principal. Si no recordáis bien estos conceptos podéis revisar la Sección 5 del contenido teórico del módulo actual.

Por lo tanto, cualquier proceso que se ejecute mediante Yarn se va a ejecutar (si lo considera necesario) utilizando todos los recursos del sistema. En este ejercicio vamos a ver como usar Yarn, como analizar las características del programa ejecutado y como ha finalizado la ejecución.

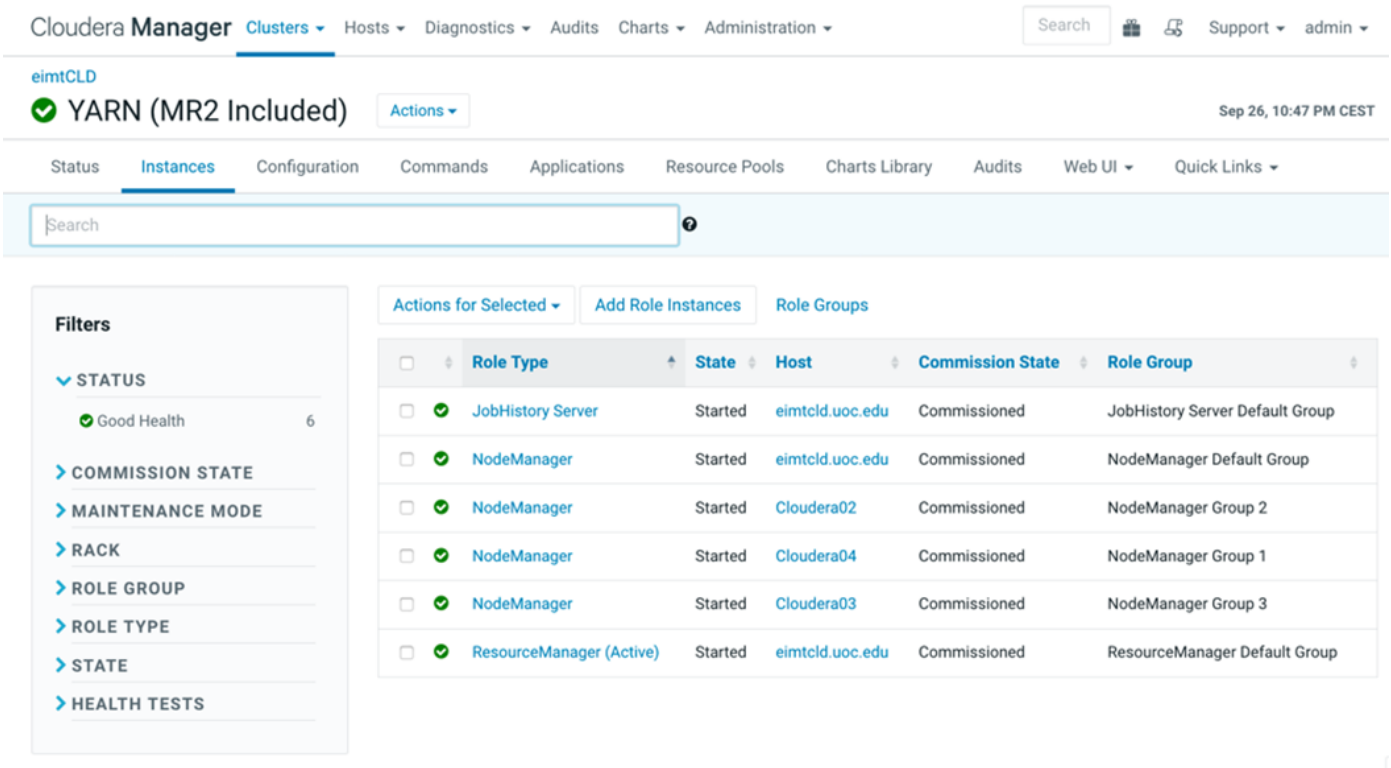


Figura 2: Configuración del gestor de recursos Yarn en la máquina usada en la asignatura.

Si revisamos la configuración del Spark Context que estábamos usando hasta ahora en los notebooks:

```
sc = pyspark.SparkContext(master="local[1]",
appName="PAC1_vuestro_loginUOC")
```

podemos ver que el parámetro `master="local[1]"` indica que el programa sólo se ejecutará usando la máquina `local[1]`. Para entender bien los parámetros de configuración, se recomienda revisar el manual de referencia[Spark Programming Guide] (https://spark.apache.org/docs/2.1.1/programming-guide.html#initializing-spark). Por lo tanto, en los ejercicios ejecutados mediante notebooks Jupyter, no estamos utilizando el gestor de recursos, sino que ejecutamos los programas solamente utilizando la máquina principal. Para configurar el Spark Context mediante el gestor de recursos solo haría falta cambiar el parámetro `master="local[1]"` por `master="yarn"`. Aun así, en esta PEC nosotros vamos a lanzar los programas directamente desde la terminal de Linux mediante el comando `spark-submit`.

2 Ejecución de Spark (2.5 puntos)

2.1 Descripción del programa SparkPi (0.5 puntos)

El este ejercicio práctico vamos a trabajar sobre un programa ya compilado y preparado para ejecutarse. El programa que vamos a usar es un ejemplo que viene incorporado en la instalación de Cloudera, se suele usar como ejemplo base para realizar tareas de comprobación de operatividad del sistema y rendimiento. Podéis encontrar el programa en:

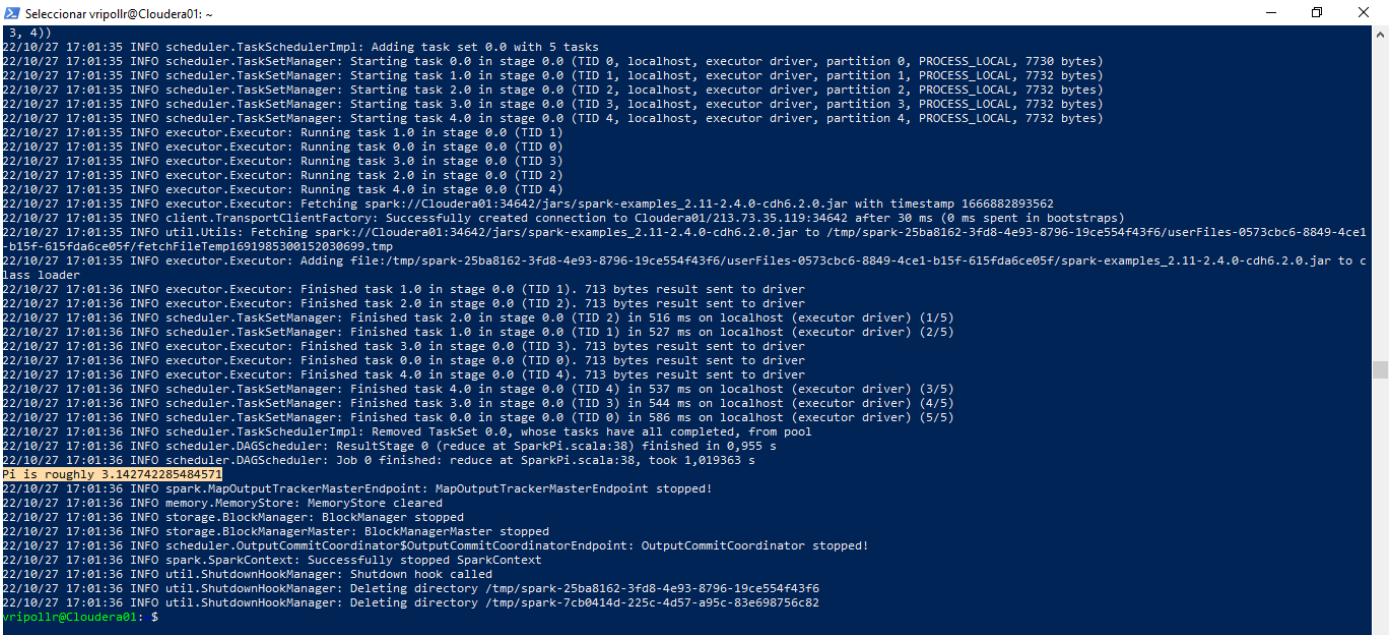
`/opt/cloudera/parcels/CDH/lib/spark/examples/jars/spark-examples_2.11-2.4.0-cdh6.2.0.jar`

Como podéis ver el programa está construido en Java, pero no vamos a modificarlo. Solo lo vamos a ejecutar en el siguiente apartado.

En este primer apartado se pide que describáis:

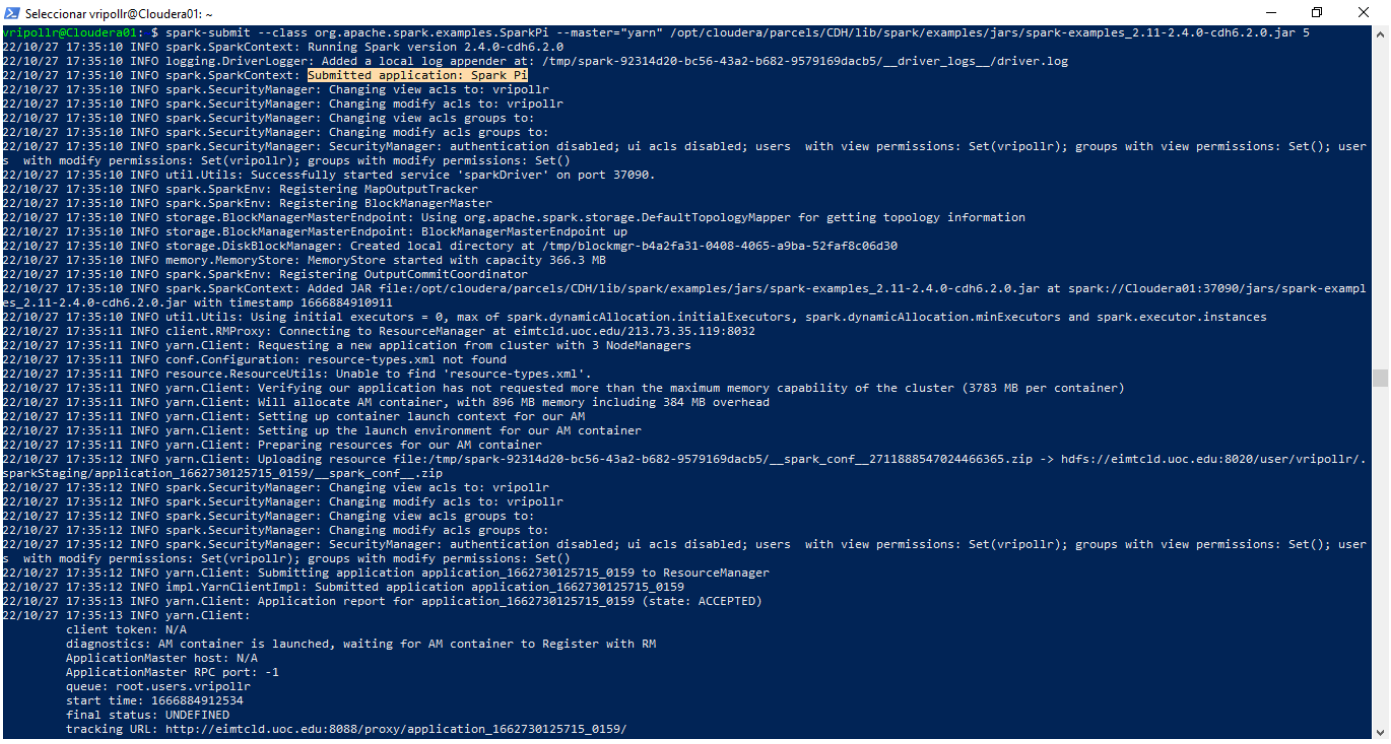
¿Qué hace el programa?

Da una aproximación del valor de pi.



¿Cómo lo hace?

Mediante la aplicación Spark Pi, que implementa una variación del problema clásico de probabilidad de la "Aguja de Bufón". Concretamente, define el cuadrado de vértices (0,0), (0,1), (1,0) y (1,1) así como la circunferencia inscrita, centrada en (0.5,0.5). El programa genera puntos en este cuadrado y cuenta cuántos de ellos caen en la circunferencia. Nótese que el área del cuadrado es 1 y la de la circunferencia pi/4. La ley de grandes números asegura que los ensayos que realiza el programa tienden al ratio entre ambas áreas, que es pi/4. Por este motivo, el programa finaliza multiplicando por 4.



El código se puede consultar en [https://github.com/apache/spark/blob/master/examples/src/main/python/pi.py (https://github.com/apache/spark/blob/master/examples/src/main/python/pi.py)](https://github.com/apache/spark/blob/master/examples/src/main/python/pi.py)

```
import sys
from random import random
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    spark = SparkSession\
        .builder\
        .appName("PythonPi")\
        .getOrCreate()

    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_: int) -> float:
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 <= 1 else 0

    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).reduce(add)
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

¿Qué parámetros recibe para su ejecución y para que sirven?

Recibe el número de puntos que se desean generar dentro del cuadrado. Este número sirve para estimar el número pi mediante el procedimento que se ha descrito. Cuanto mayor sea el número, la aproximación será más exacta.

2.2 Ejecución (1 puntos)

Revisad la documentación de Cloudera[Running Spark applications on YARN] (https://docs.cloudera.com/runtime/7.2.1/running-spark-applications/topics/spark-running-spark-apps-on-yarn.html) y ejecutad el programa usando Yarn en modo cliente, con una memoria por ejecutor de 1 gigabyte y configurando el programa SparkPi para usar 10 particiones. Debéis ejecutar el programa mediante una terminal (logeados vía ssh o directamente con el terminal de Jupyter), mediante el comando spark-submit.

Una vez ejecutado, el resultado esperado debe ser parecido al siguiente:

```
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Starting task 7.0 in stage 0.0 (TID 7, Cloudera01, executor
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 0.0 (TID 6) in 51 ms on Cloudera0
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Starting task 8.0 in stage 0.0 (TID 8, Cloudera01, executor
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 53 ms on Cloudera0
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Starting task 9.0 in stage 0.0 (TID 9, Cloudera01, executor
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 307 ms on Cloudera
20/09/19 17:31:54 INFO scheduler.TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 90 ms on Cloudera0
20/09/19 17:31:54 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/09/19 17:31:54 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 4,997 s
20/09/19 17:31:54 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 5,078599 s
Pi is roughly 3.14017514017514
20/09/19 17:31:54 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
20/09/19 17:31:54 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
20/09/19 17:31:54 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
20/09/19 17:31:54 INFO cluster.SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
 services=List(),
 started=false)
20/09/19 17:31:54 INFO cluster.YarnClientSchedulerBackend: Stopped
20/09/19 17:31:54 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/09/19 17:31:54 INFO memory.MemoryStore: MemoryStore cleared
20/09/19 17:31:54 INFO storage.BlockManager: BlockManager stopped
20/09/19 17:31:54 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
```

spark-submit --class org.apache.spark.examples.SparkPi --master yarn --executor-memory 1G --deploy-mode client /opt/cloudera/parcels/CDH/lib/spark/examples/jars/spark-examples_2.11-2.4.0-cdh6.2.0.jar 10

vripollr@Cloudera01: ~

```
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 2004 ms on Cloudera
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Starting task 8.0 in stage 0.0 (TID 8, Cloudera01, executor 1
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 2143 ms on Cloudera
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Starting task 9.0 in stage 0.0 (TID 9, Cloudera01, executor 1
22/10/27 18:00:04 INFO storage.BlockManagerMasterEndpoint: Registering block manager Cloudera01:45846 with 408
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 2032 ms on Cloudera
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 166 ms on Cloudera0
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 129 ms on Cloudera0
22/10/27 18:00:04 INFO scheduler.TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 202 ms on Cloudera0
22/10/27 18:00:04 INFO storage.BlockManagerMasterEndpoint: Registering block manager Cloudera01:39339 with 408
22/10/27 18:00:04 INFO storage.BlockManagerMasterEndpoint: Registering block manager Cloudera01:42593 with 408
22/10/27 18:00:05 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Registered executor NettyRpcEndpointRe
22/10/27 18:00:05 INFO spark.ExecutorAllocationManager: New executor 9 has registered (new total is 8)
22/10/27 18:00:05 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on Cloudera01:39708 (size:
22/10/27 18:00:05 INFO storage.BlockManagerMasterEndpoint: Registering block manager Cloudera01:35632 with 408
22/10/27 18:00:05 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 1524 ms on Cloudera
22/10/27 18:00:05 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on Cloudera01:45846 (size:
22/10/27 18:00:05 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on Cloudera01:39339 (size:
22/10/27 18:00:05 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Registered executor NettyRpcEndpointRe
22/10/27 18:00:05 INFO spark.ExecutorAllocationManager: New executor 8 has registered (new total is 9)
22/10/27 18:00:05 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on Cloudera01:42593 (size:
22/10/27 18:00:05 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Registered executor NettyRpcEndpointRe
22/10/27 18:00:05 INFO spark.ExecutorAllocationManager: New executor 10 has registered (new total is 10)
22/10/27 18:00:05 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) in 1675 ms on Cloudera
22/10/27 18:00:05 INFO storage.BlockManagerMasterEndpoint: Registering block manager Cloudera01:42241 with 408
22/10/27 18:00:05 INFO scheduler.TaskSetManager: Finished task 5.0 in stage 0.0 (TID 5) in 1621 ms on Cloudera
22/10/27 18:00:06 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 0.0 (TID 6) in 1592 ms on Cloudera
22/10/27 18:00:06 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
22/10/27 18:00:06 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 8,521 s
22/10/27 18:00:06 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 8,579657 s
22/10/27 18:00:06 INFO storage.BlockManagerMasterEndpoint: Registering block manager Cloudera01:43295 with 408
Pi is roughly 3.1404311404311405
22/10/27 18:00:06 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
22/10/27 18:00:06 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
22/10/27 18:00:06 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
22/10/27 18:00:06 INFO cluster.SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
 services=List(),
 started=false)
22/10/27 18:00:06 INFO cluster.YarnClientSchedulerBackend: Stopped
22/10/27 18:00:06 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
22/10/27 18:00:06 INFO memory.MemoryStore: MemoryStore cleared
22/10/27 18:00:06 INFO storage.BlockManager: BlockManager stopped
22/10/27 18:00:06 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
```

2.3 Modos de ejecución (1 puntos)

¿Qué diferencia hay entre ejecutar una aplicación de Spark en YARN en modo cliente o modo clúster? ¿En que casos utilizarías cada uno de estos modos?

Tal y como se explica en https://spark.apache.org/docs/1.6.2/running-on-yarn.html#configuration (https://spark.apache.org/docs/1.6.2/running-on-yarn.html#configuration), en el modo clúster el driver de Spark se ejecuta en el application master, mientras que en el modo cliente el driver se ejecuta en el proceso del cliente. En otras palabras, en modo client cuando un usuario envía un spark submit (formado por el driver program y los executor program), el driver program se ejecuta en el edge node ocupando memoria. En el modo cluster, el driver aunque es enviado al edge node, se envía a las máquinas del clúster para ser ejecutado en función de los recursos disponibles. Para trabajar en modo producción es preferible el modo clúster porque incluso si la cuenta con que se trebaja se desactiva el programa se seguirá ejecutado. En el modo cliente esto no sucede. Así, se prefiere el modo cliente en tareas menos importantes como las de testeo. Como consecuencia del funcionamiento del modo cliente, una desventaja importante de este es que si hay varias personas enviando tareas de Spark al edge node, todos los drivers correspondientes a estas tareas serán ejecutados en este mismo edge node lo que puede implicar una falta de recursos para algunos de estos drivers. En el modo clúster, los drivers no se ejecutan en el edge node sino que es el edge node el que gestiona donde tienen que ejecutarse estos drivers. Esta gestión permite distribuir los recursos de una forma más eficiente. De nuevo, en tareas de producción es preferible el modo clúster debido a que habrá menos posibilidades de encontrarse con problemas de memoria.

3 Spark History Server (5 puntos)

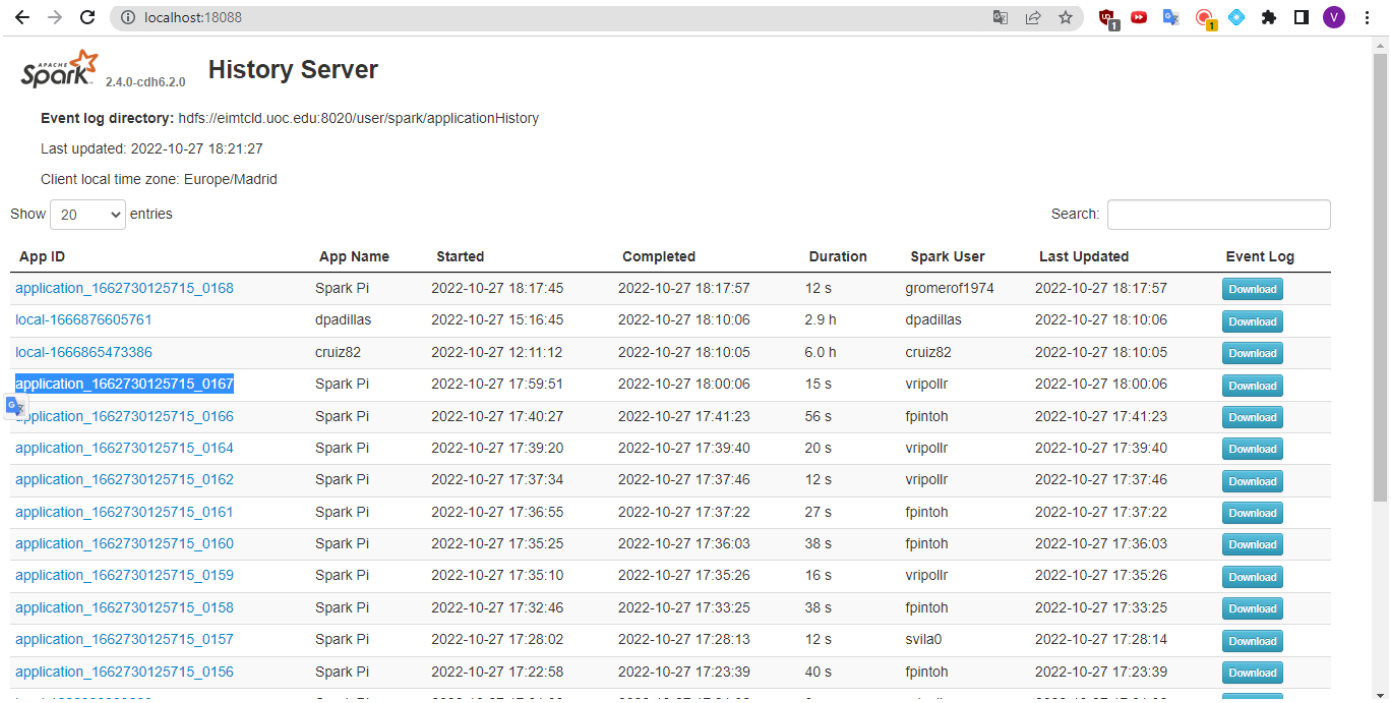
En este bloque de preguntas vamos a trabajar con la herramienta de monitorización que ofrece Spark. Entrar en la Web UI del Spark history Server[Monitoring Spark Applications] (https://docs.cloudera.com/documentation/enterprise/6/6.2/topics/operation_spark_applications.html) habilitando primero un túnel SSH:

http://localhost:18088

y contestar a las siguientes preguntas (todas las respuestas se deben acompañar de evidencias con capturas de pantalla). Revisad con cuidado la información que os da el Spark History Server ya que no solo aparecen vuestros programas.

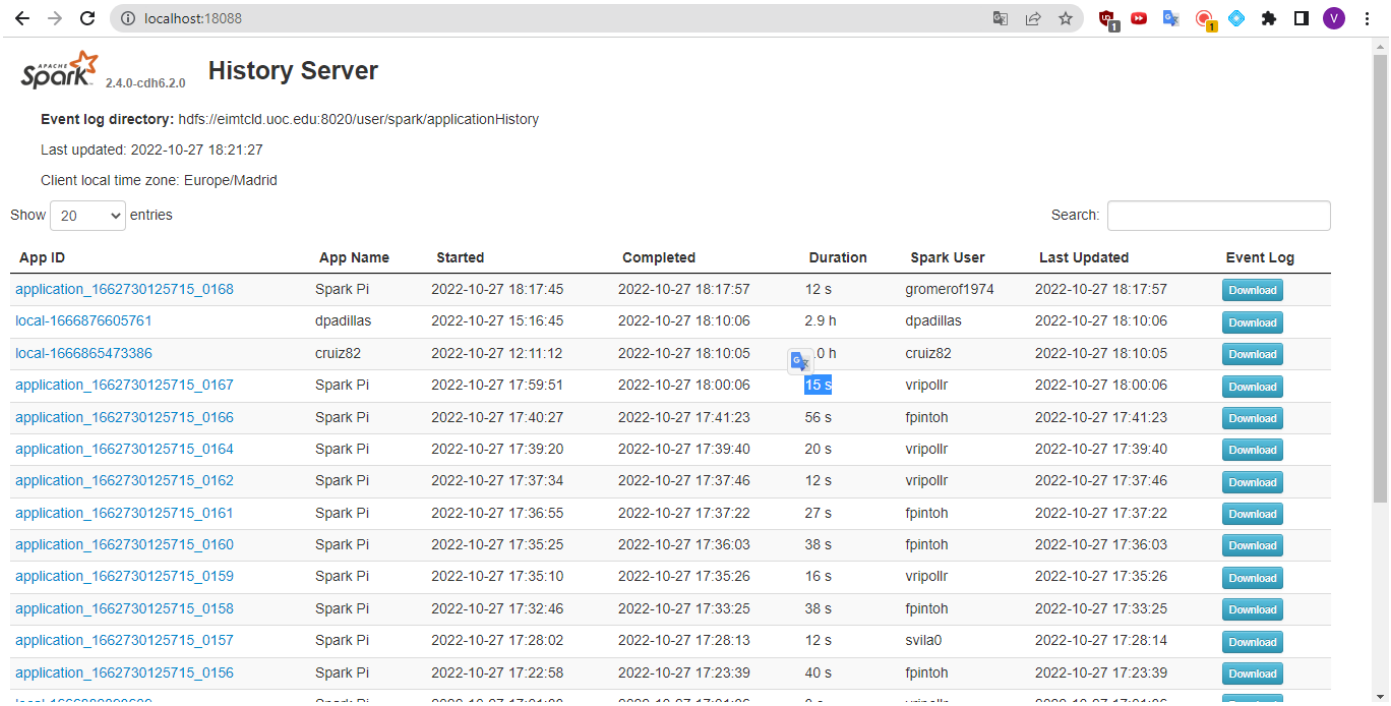
3.1 ¿Cuál ha sido el identificador de aplicación que nos ha asignado YARN cuando hemos ejecutado nuestro programa? (0.5 punto)

application_1662730125715_0167



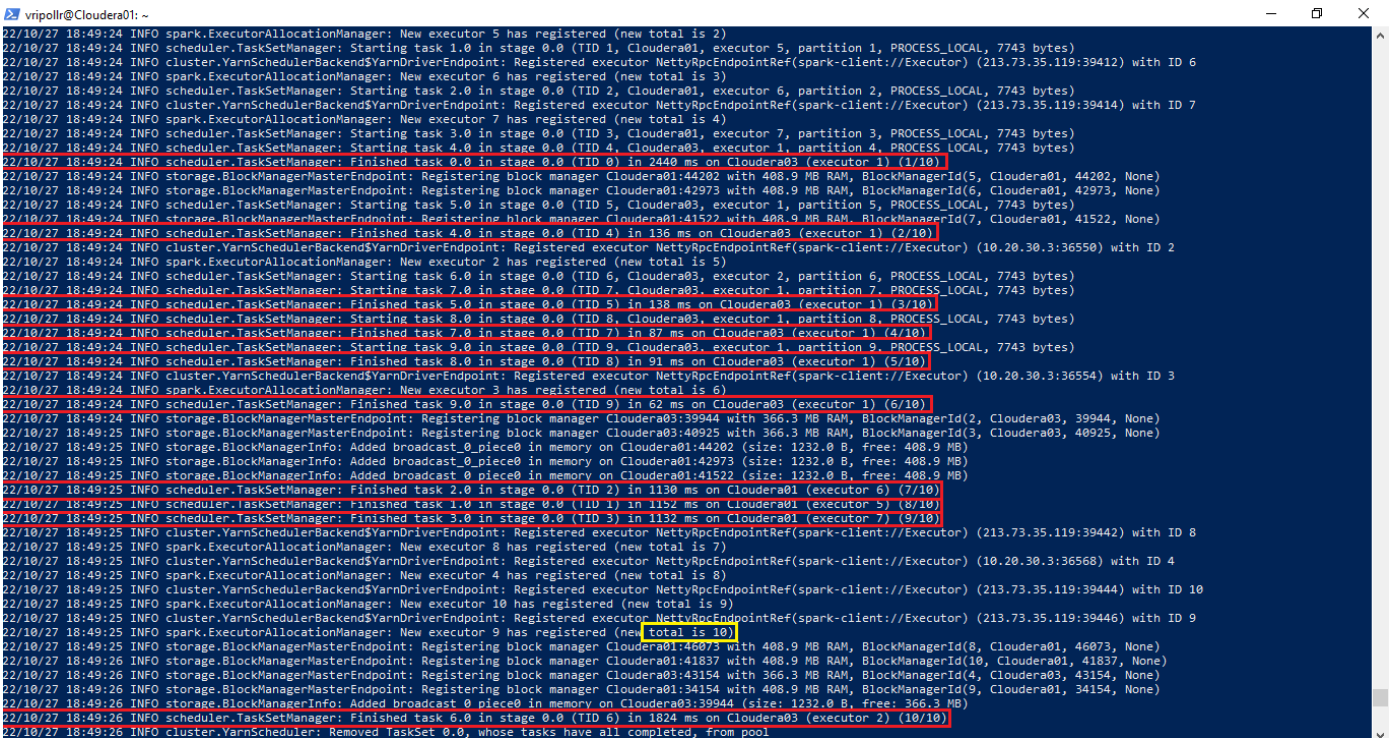
3.2 ¿Qué tiempo ha durado la ejecución? (0.5 punto)

15 segundos



3.3 ¿Cuántos executors han intervenido durante nuestra ejecución y en que nodo se han levantado? (0.5 punto)

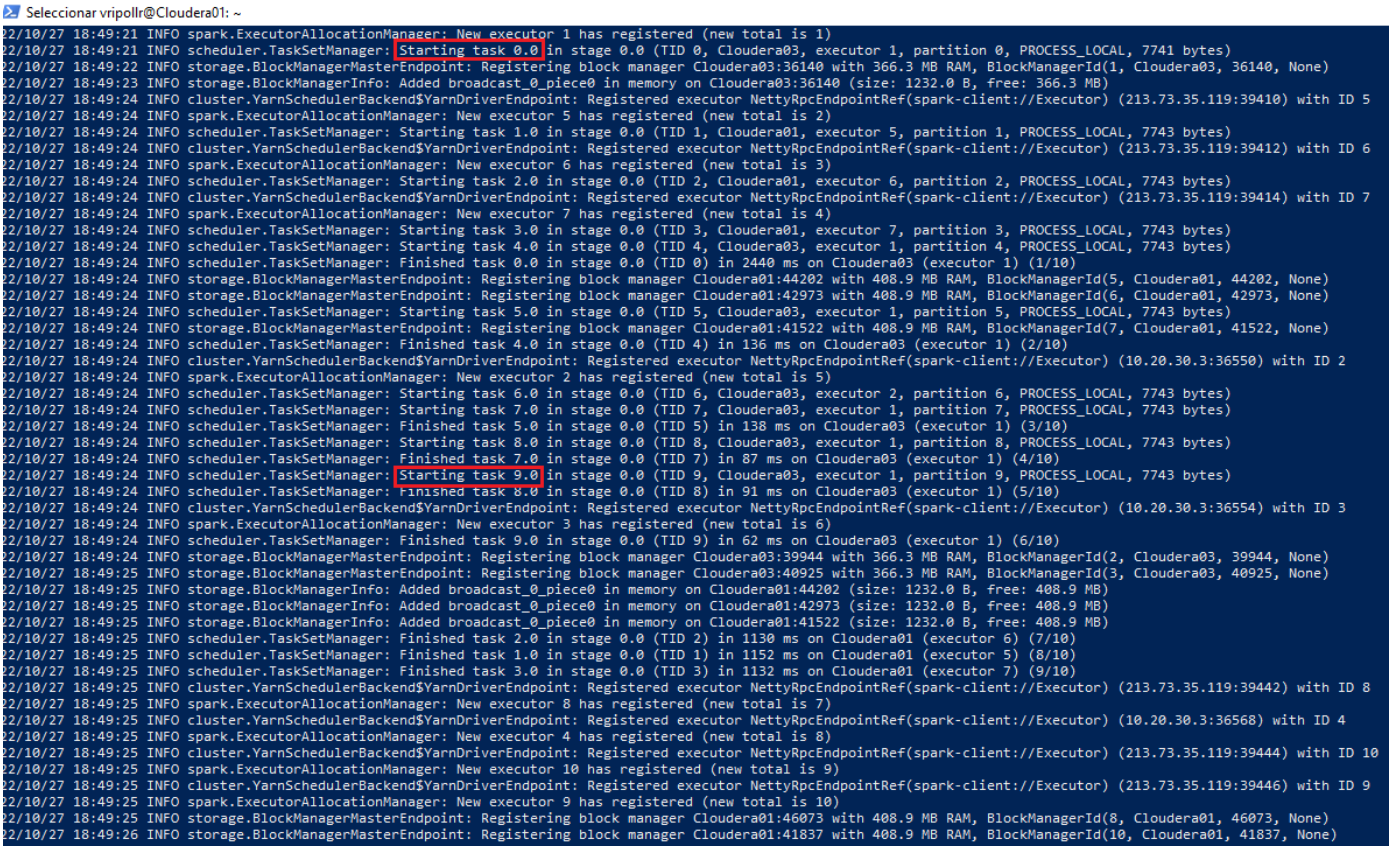
Si se ejecuta el siguiente código: spark-submit --class org.apache.spark.examples.SparkPi --master yarn --executor-memory 1G --deploy-mode client /opt/cloudera/parcels/CDH/lib/spark/examples/jars/spark-examples_2.11-2.4.0-cdh6.2.0.jar 10 Se obtiene:



Se observa que se han creado 10, pero en las 10 tasks que se querían aplicar han intervenido 5: executors 1, 2, 5, 6 y 7. El executor 1 se ha levantado en el nodo Cloudera 03 El executor 2, en Cloudera 03 El executor 5, en Cloudera 01 El executor 6, en Cloudera 01 El executor 7, en Cloudera 01

3.4 ¿Cuántas Tasks se han ejecutado para nuestro job? (0.5 punto)

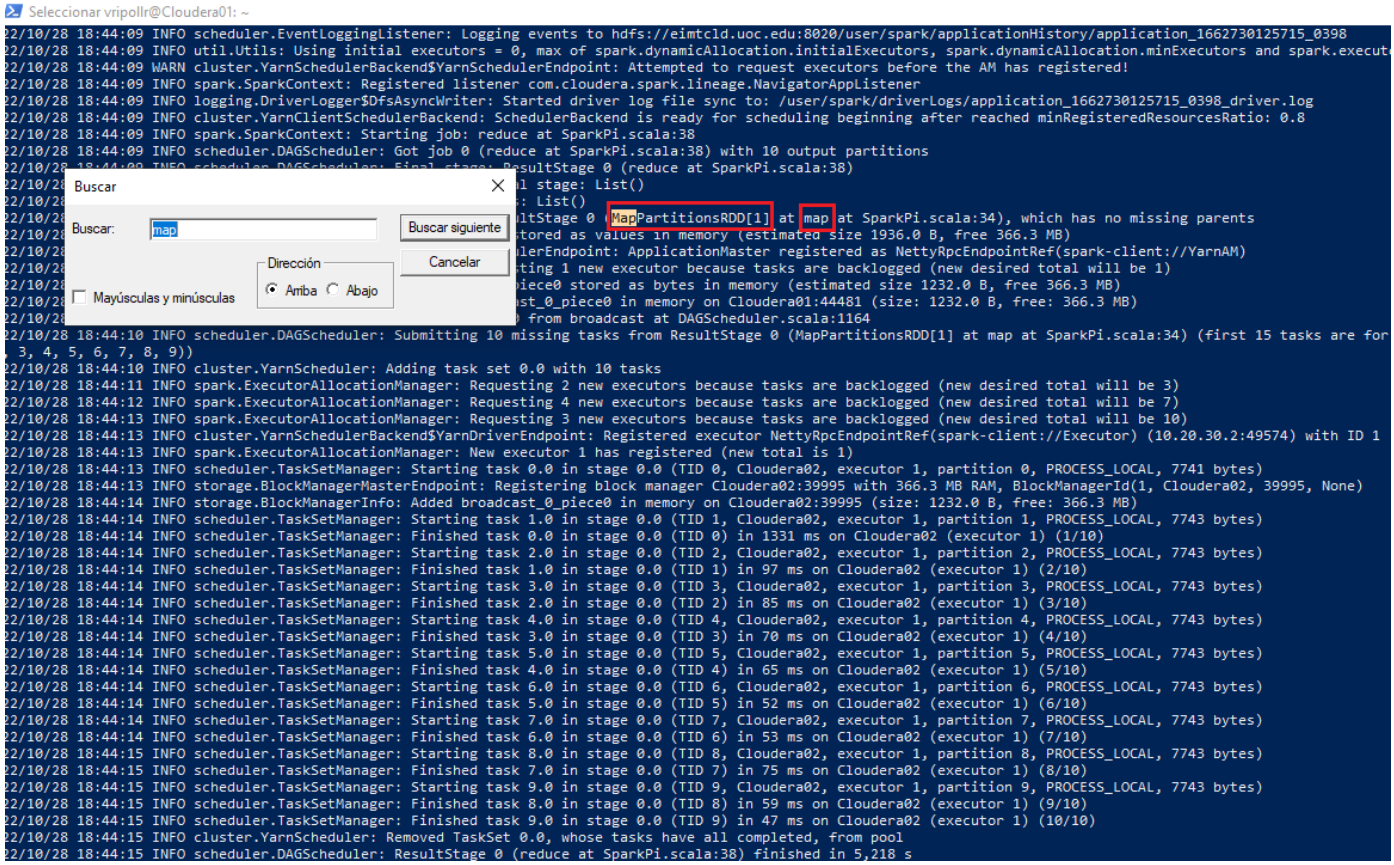
Se ejecutan 10. Se observa que el TaskSetManager gestiona el inicio y el fin de 10 tasks: task 0.0, 1.0,..., 9.0.



3.5 ¿Cuántas transformaciones se han invocado durante nuestro programa? (1

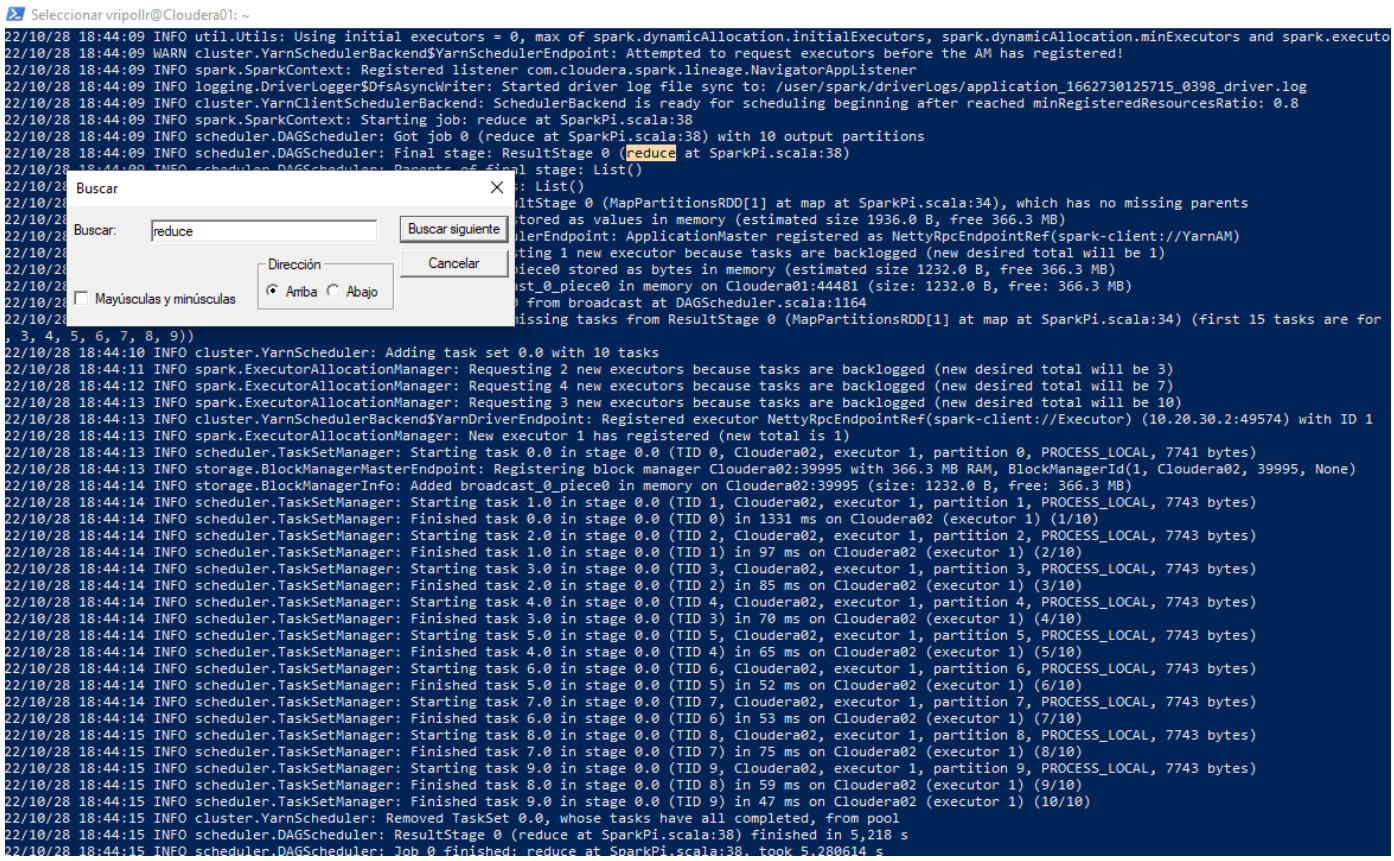
punto)

Si se consulta el listado de transformaciones en https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations (https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations) y se utiliza el comando ctrl+f dentro del terminal para buscar coincidencias, se encuentran únicamente transformaciones de tipo map():



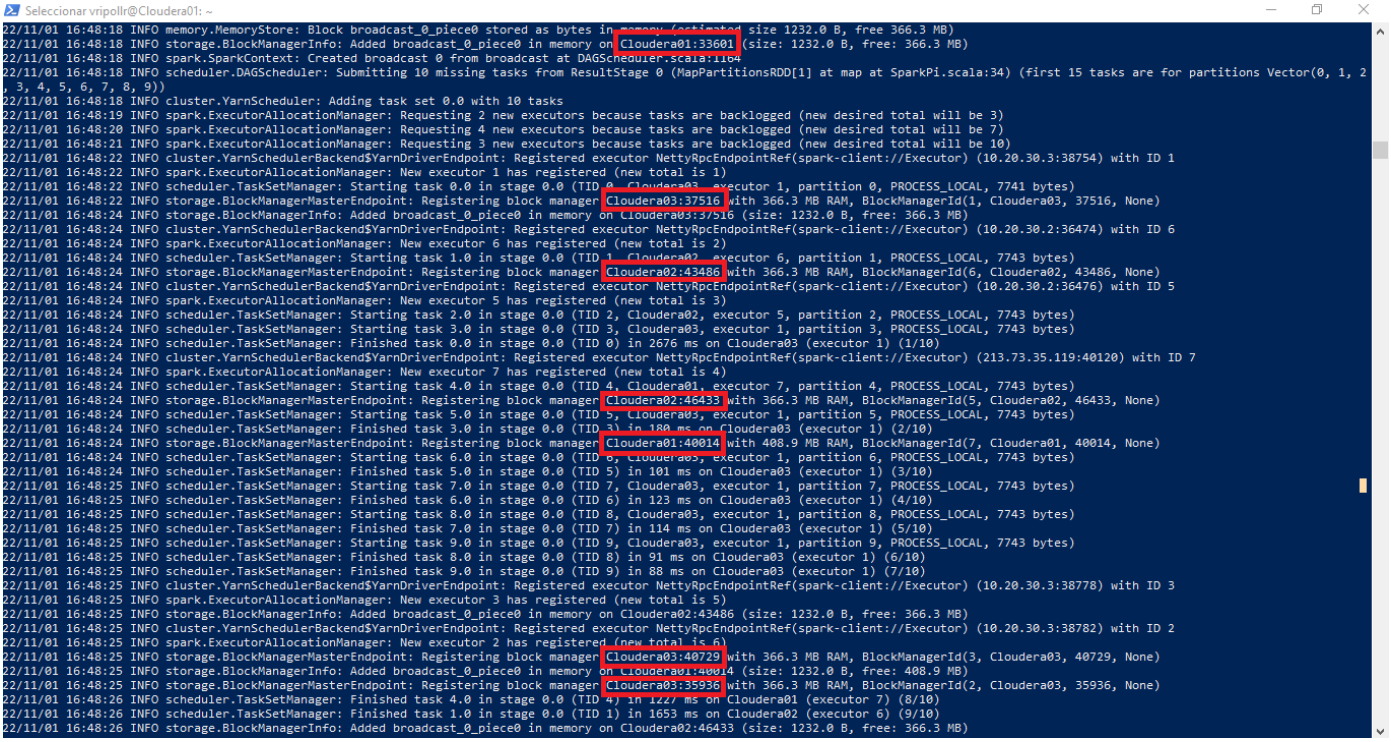
3.6 ¿Cuántas acciones se han invocado durante nuestro programa? (1 punto)

Si se consulta el listado de acciones en https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions (https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions) y se utiliza el comando ctrl+f dentro del terminal para buscar coincidencias, se encuentra únicamente la acción reduce():



3.7 ¿Cuántos cores totales de nuestro clúster han intervenido en nuestra ejecución? (1 punto)

Intervienen 8: Cloudera01:33601, Cloudera01:40014, Cloudera01:38524, Cloudera02:43486, Cloudera02:46433, Cloudera03:40729, Cloudera03:35936 y Cloudera03:37516.



4 Gestión de aplicaciones en YARN (2.5 punto)

Como ya sabéis, YARN es el encargado de manejar los recursos de los nodos que forman el clúster de manera global y es él quien tiene la responsabilidad de distribuir las aplicaciones para que se ejecuten de manera paralelizada y balancear la carga entre ellas.

Una vez enviada una aplicación a YARN en modo cluster, esta se ejecuta bajo su control, por eso, si queremos cancelar una aplicación, no basta con hacer un `cntr+c` sino que debemos indicárselo a YARN.

Para esta tarea y muchas otras, tenéis disponible el comando `yarn` en la terminal[YARN Commands](https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YarnCommands.html). En este ejercicio vamos ver algunas de las posibilidades que ofrece.

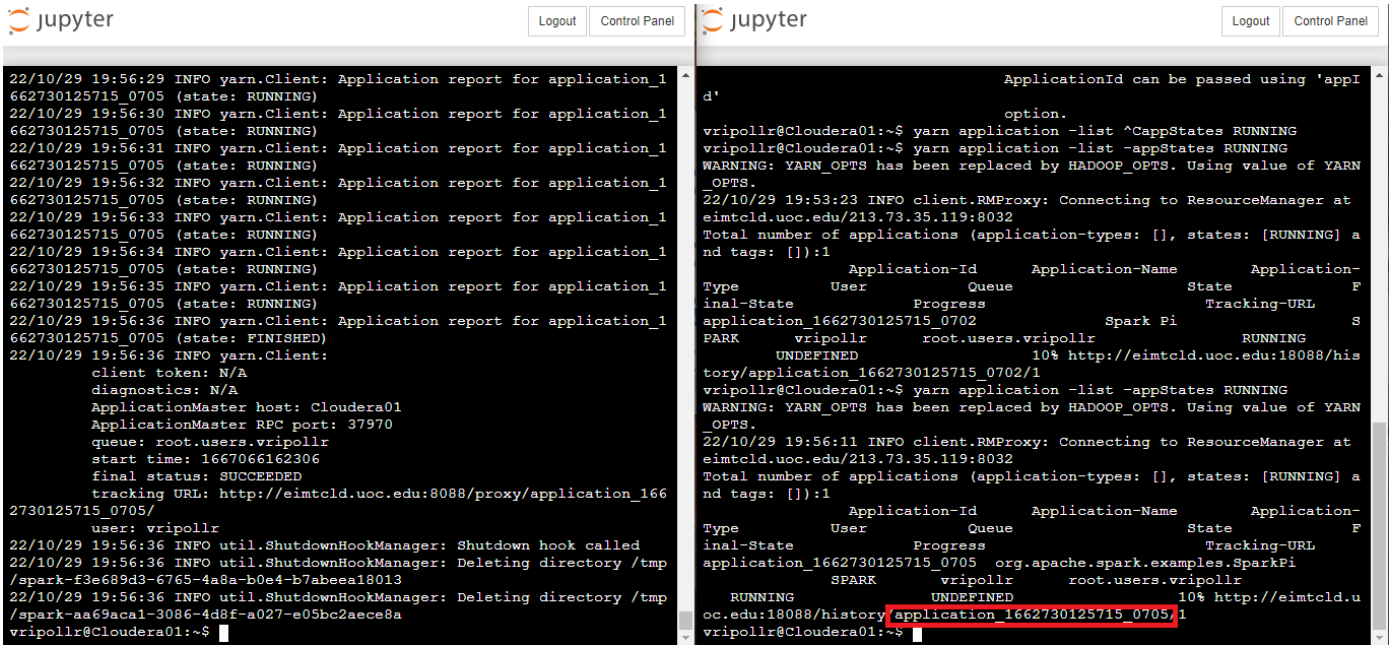
Todas las respuestas a las preguntas siguientes deben acompañarse de evidencias con capturas de pantalla.

Para este ejercicio, vamos a abrir dos terminales dentro del JupyterLab, uno en cada pestaña. En el primero, volveremos a ejecutar la aplicación SparkPi en modo clúster incrementando el número de muestras de 10 a 10000.

En el segundo, y mientras SparkPI se está ejecutando, debéis consultar las aplicaciones que YARN está ejecutando mediante el comando `yarn`.

¿Cuál es la ID que se ha asignado (1.25 puntos)?

En la primer pestaña se ha aplicado el comando spark-submit --class org.apache.spark.examples.SparkPi --master yarn --executor-memory 1G --deploy-mode cluster /opt/cloudera/parcels/CDH/lib/spark/examples/jars/spark-examples_2.11-2.4.0-cdh6.2.0.jar 10000 En la segunda, mientras se ejecutaba el anterior, se ha aplicado el comando yarn application -list -appStates RUNNING



El ID asignado es application_1662730125715_0705

Una vez obtengáis la ID, mientras nuestra aplicación esta todavía en ejecución (tarda 40 segundos en ejecutarse), utilizad el comando `yarn` para parar su ejecución.

¿Qué comando habéis ejecutado (1.25 puntos)?

Para parar la aplicación se puede usar el comando yarn application -kill <ID de la aplicación>. Nótese que en cada spark-submit la ID cambia, en la siguiente imagen primero se ha tenido que aplicar el comando yarn

application -list -appStates RUNNING para conocer la ID actual y luego finalizarla con yarn application -kill application_1662730125715_0712: