CREACIÓN DE UNA PLATAFORMA BIG DATA PARA INVESTIGADORES

UNIVERSIDAD POLITÉCNICA DE VALENCIA

MÁSTER EN BIG DATA ANALYTICS 2015-2016

Trabajo de Fin de Máster

Antonio Eslava Polo

Dirección

Dr. Francisco M. Rangel Pardo

1 INTRODUCCIÓN

El reciente desarrollo de tecnologías que manejan datos en cantidades hasta ahora inabordables, abre oportunidades a ciencias que habían encontrado techos técnicos en los sistemas utilizados hasta el momento.

Así, la Inteligencia Artificial o el Análisis de Negocio (*Business Intelligence*) viven un nuevo empuje con la aplicación de tecnologías como la computación en clúster y la paralelización.

Tampoco las ciencias sociales, como potencial usuario de datos en grandes cantidades, quedan aparte de esta corriente. Por una parte, la información que proporcionan las redes sociales (Twitter, Facebook), así como la recogida por las empresas que proporcionan servicios (telefonía), e incluso la ofrecida por los gobiernos en datos abiertos, se producen en elevadas cantidades.

Ante esta situación, un investigador puede encontrarse con grandes conjuntos de datos y la necesidad de analizarlos para obtener las conclusiones a que aspira.

1.1 PROBLEMAS PARA LOS INVESTIGADORES

Uno de los elementos que dificultan la generalización de este avance es que las tecnologías de Big Data son relativamente costosas o complejas. La aproximación tradicional de grandes sistemas monopuesto con aplicaciones complejas (SAS, Oracle) limita el número de investigadores a aquellos que pueden acceder a estos sistemas. La aproximación reciente, la englobada en el concepto "Big Data", es aún compleja. Por una parte, es económico el uso de clústeres de pago por uso durante cortos periodos de tiempo, con lo que se evita realizar inversiones en infraestructura. Pero por otra, el aprovechamiento de estos clústeres obliga a unos conocimientos técnicos que desvían al investigador de su objetivo y competencias fundamentales.

El objeto de este trabajo es proporcionar una plataforma técnica que permita a un investigador aprovechar la computación paralela en nube, con una mínima inversión, tanto económica, como en conocimientos técnicos ajenos a los propios de su disciplina de estudio.

1.2 ESTRUCTURA DEL TRABAJO

El objetivo se ha dividido en dos bloques principales de tareas, creación de la plataforma y aplicación a un caso de uso:

1. Creación de la plataforma

De modo automático, se crea un grupo de ordenadores virtuales en la nube, y se conforman como un clúster. Con esto, el investigador ya dispondría de la plataforma básica. Pero necesitaría conocimientos de paralelización y clúster para poder aprovechar los entornos de Hadoop y Spark instalados.

La plataforma creada tiene dos características principales: utilización de la nube de alquiler y creación de clústers para computación en paralelo.

La **computación en nube** de alquiler permite fundamentalmente la utilización sólo durante los procesos de cálculo, generando costes únicamente durante el periodo en que se encuentra en uso. La herramienta es capaz de crear los sistemas necesarios en nube, ponerlos en marcha y eliminarlos una vez realizados los cálculos.

La **computación en paralelo** permite multiplicar la potencia de cálculo según las necesidades, sobrepasando así los límites propios de un solo sistema. La herramienta es capaz de aprovechar la división de la potencia de cálculo entre varios ordenadores, de modo automático.

2. Aplicación a un caso de uso

Además de la automatización del clúster, se proporciona un caso de uso. Para casos como el propuesto, la plataforma puede ser aprovechada de una manera cerrada y completa, sin necesidad de ninguna modificación. El caso propuesto es muy frecuente en varias áreas de investigación: **la obtención de coeficientes de correlación entre líneas de una matriz**.

Si la computación en paralelo fuera sencilla, bastaría sólo con crear la herramienta y ponerla a disposición del investigador.

En cambio, los sistemas de paralelización (Hadoop y Spark en este caso y, en general, cualquier otro similar) requieren capacidades nuevas de programación para su uso.

Para minimizar el coste de aprovechamiento de estas capacidades, se ha optado por crear un caso de uso. Se proporciona con él una especie de "plantilla" o "equeleto", que puede ser fácilmente modificado para adaptarlo a otras necesidades similares.

Este caso es de general utilidad en muchos campos de investigación: el cálculo de correlación estructural entre elementos de una matriz dada. Para ello, en el ejemplo propuesto, se ha utilizado el coeficiente de correlación de Pearson, pero cualquier investigador con básicos conocimientos del lenguaje Python podrá adaptar el programa a sus propios procedimientos de cálculo.

Mediante esta adaptación particular, el investigador generalizaría el caso de uso propuesto, adaptándolo a sus necesidades particulares, obteniendo provecho, de manera sencilla, del clúster creado en la primera fase.

Con la inclusión de esta parte, el sistema completo es capaz sucesivamente, y de modo automático, de:

- Recibir una matriz de unos y ceros, compuesta por varios miles de filas y columnas.
- Crear y configurar un clúster suficiente.
- Efectuar el cálculo de la correlación estructural.
- Depositar en un sistema externo una matriz resultado con los coeficientes de correlación.
- Destruir todos los elementos del clúster, evitando incurrir en más costes de los estrictamente necesarios.

2 PLATAFORMA DE INFRAESTRUCTURA EN NUBE

La herramienta de paralelización en nube consta de un grupo de ordenadores virtuales, denominados "instancias". Estas instancias se relacionan entre sí formando grupos denominados "clúster", que son capaces de coordinar las actuaciones que realiza cada uno en paralelo.

La plataforma de infraestructura en nube está compuesta por dos niveles de elementos:

a) Las instancias, que son creadas y destruidas de modo automático.

b) Los sistemas de clúster, que son instalados en las instancias y configurados para que actúen solidariamente.

2.1 ELEMENTOS UTILIZADOS

Los sistemas de computación en nube ofrecen una amplia variedad de regímenes de uso: alquiler o propiedad, tipo de elemento a alquilar (sistemas enteros o servicios de cómputo), capacidad de los elementos a utilizar, tipo de sistema, etc.

Para esta herramienta, se han seleccionado los siguientes elementos:

2.1.1 Proveedor de nube: Amazon AWS[1].

Es el pionero en este entorno y ha mantenido el liderazgo desde su creación. En el momento de realización de este trabajo, otros fabricantes como Microsoft[2], Google[3], IBM[4] u Oracle[5], están desarrollando su oferta en nube a gran velocidad, por lo que, con toda probabilidad, los conceptos desarrollados aquí puedan ser perfectamente portados a sus entornos. La elección de AWS se ha basado en la madurez y conocimiento generalizado de la herramienta, tanto a nivel técnico como de costes.

2.1.2 Servicio en nube: Ec2

Es el servicio de AWS que proporciona ordenadores virtuales completos. Existen otros servicios, como EMR (Elastic Map Reduce), que proporcionan un clúster Hadoop o Spark ya instalado, ocultando al usuario la complejidad de la creación y mantenimiento de éste. En la elección del servicio fue considerada esta alternativa. Finalmente, la decisión se decantó por el uso de instancias puras, creadas ex profeso, para disponer de mayor control sobre ellas y una mayor posibilidad de parametrización. Por otra parte, el servicio EMR utiliza, como mínimo, instancias de un tipo relativamente costoso (*xlarge*); por lo que se consideró interesante explorar la posibilidad de instancias más económicas.

2.1.3 Sistema Operativo: Ubuntu

En concreto, se utiliza el sistema Ubuntu 14.04 LTE, ofrecido por AWS. La elección de un sistema Linux obedece a que es éste el más ampliamente difundido para la mayoría de las herramientas utilizadas en este entorno. Dentro de Linux, la elección

[1] https://aws.amazon.com/es/
[2] https://azure.microsoft.com/es-es/
[3] https://cloud.google.com/
[4] https://www.ibm.com/cloud-computing/es/es/
[5] https://cloud.oracle.com/home

de Ubuntu ha obedecido a su disponibilidad en la plataforma y su universalidad. Salvo elementos menores de sintaxis, no es ésta una elección fundamental, pudiendo el sistema ser portado a otras plataformas Linux con muy poco coste. En el momento de la publicación, la recomendación es sustituir esta versión por la 16 LTE, a causa de problemas de seguridad. Al haber sido realizado el sistema con la versión anterior, se ha decidido mantenerla a lo largo del documento.

2.1.4 Sistema de clúster: Hadoop / Spark

Los sistemas de cálculo de paralelización en clúster necesitan dos niveles:

a) El gestor del clúster, que distribuye y coordina los trabajos en los nodos.

b) el gestor de cálculo, que realiza la computación necesaria, dividiendo la tarea y entregando las partes al gestor del clúster.

Como producto, Hadoop incluye los dos elementos: el gestor de clúster (YARN) y el gestor de cálculo (Map Reduce). Spark, por su parte, sólo incluye el gestor de cálculo, utilizando un gestor de clúster externo, como YARN o MESOS.

La combinación utilizada en la herramienta ha sido Hadoop + Spark, de modo que la gestión del clúster es realizada por YARN, y el cálculo, mediante Spark.

2.1.5 Sistema de creación de las instancias: Script "ad hoc"

La creación de instancias en nube dispone de una tecnología en expansión con sistemas muy sofisticados, que permiten el despliegue, aprovisionamiento y control de instancias en nube, de modo automatizado. Éstos van desde el sistema propio de AWS, *Cloud Formation*, hasta los ampliamente generalizados como *Puppet*, *Ambari* o *Jenkins*, alternativas potentes y maduras. En este caso, debido a criterios de simplicidad y uso efímero, se ha optado por realizar la creación de instancias y su integración en clúster mediante un script propio que, al mismo tiempo, realiza el lanzamiento de los cálculos.

2.1.6 Lenguaje de programación: Python.

Para el lanzamiento y control de los procesos, se ha utilizado un *script* de Python 3.x. La elección de este lenguaje obedece a su amplia difusión en la comunidad científica. Debido a su relativa sencillez, la curva de aprendizaje de éste idioma es de bajo perfil, evitando que el investigador distraiga sus recursos, de tiempo y capacidad, en el aprendizaje de idiomas más complejos. De este modo, en caso de necesitar alguna modificación en el sistema, el investigador puede realizarla por sí mismo con menor coste. Aunque en estos momentos, conviven las versiones 2.x y 3.x de Python, las mínimas diferencias entre ellas no afectan al *script*, que puede ser ejecutado, casi sin modificaciones, en un entorno Python 2.x.

Una alternativa hubiera sido la utilización de scripts *Shell* de sistema operativo. Se ha preferido un script de Python por la mejor legibilidad para los investigadores, por ofrecer un aspecto más monolítico y menos fragmentado, y por proporcionar un único centro de control en el ordenador que lanza el proceso. Además, los *scripts* de lanzamiento de instancias diferirían según este ejecutaran en Windows, Linux o Mac, con lo que se obtiene una mayor universalidad al utilizar el idioma elegido.

2.1.7 Bibliotecas de programación no estándares: Paramiko, Boto3.

Por sencillez, se ha evitado al máximo el uso de bibliotecas de programación no incluidas en Python de modo estándar. Las dos únicas utilizadas han sido Boto3[6] y Paramiko[7]. La primera permite la creación, control y eliminación de instancias de EC2, de modo programático; mientras que Paramiko es un sistema potente y versátil, de gran implantación, que permite el acceso remoto a los sistemas, mediante el protocolo SSH. Este acceso permite la copia remota de archivos y la ejecución de órdenes, y ha sido ampliamente utilizado para el lanzamiento de todos los procesos tras la creación de las instancias. La instalación de ambas bibliotecas es muy sencilla y no representa coste de aprendizaje para un usuario mínimamente conocedor de Python.

2.2 CRITERIOS DE DISEÑO

Los principales criterios de diseño para la solución propuesta han sido: 1) Economía, 2) Simplicidad, 3) Máximo control y adaptabilidad

2.2.1 Economía

La estructura a utilizar nace y muere con cada utilización. Es, por definición, efímera. El principal rasgo de esta opción es la economía de costes monetarios, pues el alquiler por tiempo de las instancias limita los gastos de infraestructura.

Un segundo rasgo importante es que el carácter efímero permite configuraciones de seguridad básicas, sin necesidad de mayor sofisticación, como se razona a continuación. Esto ha permitido ahorros de coste el diseño y creación, al no tener que cumplir con los requerimientos más costosos de una instalación estable.

Para minimizar la complejidad, se ha optado por utilizar un sistema de acceso a las instancias mediante usuario y clave, conocidos y fijos. Esta opción, que sería inaceptable en un entorno mínimamente estable, es posible en éste. Por una parte, el acceso a las instancias sólo puede ser efectuado durante el tiempo en que duran los cálculos, o la creación de aquellas. Además de este factor tiempo, un acceso ilegítimo debería contar con el conocimiento de la dirección IP o DNS de estas instancias. Tanto las direcciones pública o privada, como el nombre DNS de cada instancia, son creados cada vez que se inicia el proceso, por lo que un atacante no dispondría del tiempo necesario para poner en marcha los sistemas de intrusión. Además, estas intrusiones deberían pasar el filtro aplicado por el proveedor, AWS, en la red. Éste, si bien permite accesos a las instancias según las reglas que defina el usuario, monitoriza comportamientos sospechosos en la red, como el rastreo de puertos.

2.2.2 Simplicidad

Dos son los principales rasgos que cumplen con este criterio: el uso de una imagen prediseñada y la utilización de configuraciones mínimas.

Uso de imagen prediseñada

[6] https://github.com/boto/boto3
[7] http://www.paramiko.org/

Elemento fundamental de la arquitectura propuesta es el uso de una imagen base, a partir de la cual clonar los nodos del clúster.

Cuando una instancia es creada *ex novo*, contiene únicamente el sistema operativo por defecto, Ubuntu en este caso. A este sistema operativo inicial hay que aplicarle configuraciones de acceso (usuario/clave) e instalarle los dos conjuntos de software utilizados, Hadoop y Spark. Además, es necesario realizar ciertas configuraciones en estos conjuntos de software para su utilización básica, e instalar diversos softwares auxiliares.

Una opción, la utilizada por los sistemas de recetas como *Cloud Formation*, de AWS, es la de crear un *script* que instale y configure estos sistemas de forma automática, cada vez que se crea una instancia nueva. En la solución propuesta, se ha optado por crear una primera instancia en la que instalar todo el software necesario, y realizar su configuración básica. Esta instancia se guarda en la forma que denomina AWS una *imagen AMI*. El *script* de creación de instancias del clúster da nacimiento a éstas a partir de aquella AMI.

El proceso es así mucho más rápido y simple de programar, pues es posible realizar el tedioso proceso de instalación y configuración de modo manual y en una sola vez, con el consiguiente ahorro de tiempo de programación. Además, la creación de instancias es mucho más rápida, pues no tiene que repetir, en cada una, un largo proceso de instalación y configuración desde cero.

Esta imagen AMI es controlada por su creador, bien el propio utilizador del sistema, que la genera para sí mismo, con las particularidades que cree convenientes; o bien por un creador diferente, que la pone a disposición del público en AWS, como el caso del autor de este documento.

El uso de esta imagen es, además, útil para la actualización a nuevas versiones, o la introducción de nuevas funcionalidades o software. Basta con la creación de una instancia a partir de la imagen existente, la actualización de aquella, o la inclusión de nuevo software, y archivar el sistema resultante en forma de nueva imagen actualizada.

Al ser la imagen un elemento estático, bajo el control de la cuenta AWS de su creador, no es susceptible de ser atacada para introducir en ella modificaciones ocultas con intención dolosa. El propio procedimiento de creación de la misma hace que la imagen nunca pueda ser modificada. En caso de crear, a partir de ella, otra imagen con modificaciones malintencionadas, el identificador de ésta sería diferente y no reconocido por el autor o la comunidad de usuarios.

Utilización de configuraciones mínimas

Aparte del propio sistema operativo, los elementos software utilizados, Hadoop y Spark, disponen de un amplio conjunto de opciones, disponibles en ficheros de configuración y parámetros de invocación.

Las técnicas de optimización de estos sistemas son muchas y variadas, pudiendo complicar en gran medida el proceso de instalación, configuración y explotación; además, la elección de los parámetros depende de muchos factores. En la mayoría de los casos, sólo es posible determinar de modo empírico la idoneidad de un conjunto

de parámetros, tras realizar pruebas y mediciones con variaciones controladas de algunos de ellos.

Teniendo en mente la sencillez de creación y uso del sistema propuesto, se ha optado por utilizar las configuraciones por defecto en la medida en que ha sido posible. Sólo se han realizado las modificaciones básicas que proporcionaban una mejora significativa del funcionamiento.

2.2.3 Control y adaptabilidad

Aunque el sistema se ofrece cerrado para su uso, con apenas necesidad de adaptación por parte del usuario, las posibilidades que ofrece han aconsejado minimizar las partes ocultas, no modificables por el usuario. Así, según el grado de conocimiento y necesidad, éste puede ser modificado para atender a necesidades diferentes a las del caso de uso que se ofrece, el del cálculo de correlación estructural entre elementos de una matriz. Asimismo, se ha considerado importante que el usuario tenga una visión continua del proceso, para verificar visualmente que se completa correctamente, haciéndolo en el tiempo y forma esperados.

Para ello, el sistema ofrece tres factores: uso de software estándar, uso de elementos abiertos y despliegue supervisado.

*2.2.3.1 **Uso de software estándar***

Por estándar se entiende aquí un software con la suficiente distribución en la comunidad técnica como para no requerir una inversión específica en sistemas con poco recorrido. La búsqueda de economía ha llevado también al uso de software abierto. Los elementos elegidos, Ubuntu, EC2 de AWS, Hadoop, Spark, Python, Paramiko, Numpy, cumplen con estos requisitos. En alguno de ellos, como Python, concurre además la sencillez de uso en comparación con otras alternativas.

*2.2.3.2 **Uso de elementos abiertos***

Por elementos abiertos se entiende aquellos en los que es posible actuar por parte del usuario. Un ejemplo de esta elección se da en la alternativa entre el servicio EMR de AWS y la creación de un script propio de creación de instancias EC2. En el primer caso, como ya se ha comentado, AWS oculta toda la estructura del clúster y ofrece sólo la opción del lanzamiento del script de paralelización. También ofrece la posibilidad de "inyectar" bibliotecas de programación y otras opciones de software en forma de parámetros. Pero lo hace de un modo poco transparente, que puede dejar "atrapado" al usuario con poca información, en caso de intentar realizar alguna modificación que mejore o adapte el funcionamiento del clúster.

El uso de una estructura creada *ex profeso*, y sobre la que se tiene control completo, permite reaccionar con más facilidad a modificaciones o problemas que otra más compacta, pero más oculta.

*2.2.3.3 **Sistema de despliegue supervisado***

La creación del clúster y el lanzamiento de la tarea de cálculo, se realizan mediante un script de Python. Este script, desde su inicio hasta su finalización, se encuentra asociado a una sesión en el ordenador desde el que se lanza. Una vez lanzado, es completamente automático y no es posible realizar más acción con él que la detención mediante la orden de *Ctl+C*. Hubiera sido posible la opción de lanzar, en la instancia máster, de modo no supervisado, los procesos de cálculo, y vigilarlos por separado

mediante la conexión a ésta. Puesto que esto comportaría más procesos en paralelo, que obligarían al investigador a aumentar las herramientas en uso, se ha preferido que el propio script vaya informando continuamente sobre el desarrollo de los procesos. Es así posible, con una sola mirada, realizar el seguimiento y estar atento a cualquier incidencia que se produzca, reaccionando ante ella. En un desarrollo posterior, sería relativamente sencillo crear una opción no supervisada; la ventaja de esta opción es que el proceso no bloquearía el ordenador desde el que se ha realizado el lanzamiento, ni dependería de que la conexión de éste con la nube se mantuviera constante y fiable.

Además, la pantalla de seguimiento del proceso proporciona una completa información sobre las instancias, para poder hacer un seguimiento de las mismas, especialmente en caso de fallo.

También informa del desarrollo de las tareas Spark, las copias de archivos entre el clúster y el ordenador local, y en general, del resultado de cualquier paso realizado, e incluso los tiempos invertidos.

```
>>>
 RESTART: Tfm\Bouncy\createCluster.py

Instance:    1
InstanceId:  i-07f7b79af0f594ee6
Private IP:  172.31.0.26
Public DNS:  ip-172-31-0-26.ec2.internal
Instance starting,  0 seconds
Instance starting,  5 seconds
Instance starting, 10 seconds
Instance starting, 15 seconds
Instance started
Public IP:   54.90.248.103

Instance:    2
InstanceId:  i-0c0651dbc7c7a65ff
Private IP:  172.31.8.137
Public DNS:  ip-172-31-8-137.ec2.internal
Instance starting,  0 seconds
```

Muestra de la pantalla de seguimiento

3 CASO DE USO. CÁLCULO DE MATRIZ DE CORRELACIÓN

La plataforma creada permite cualquier operación de cálculo programable en Python para Map Reduce de Hadoop, o Spark. Bastaría con modificar los scripts incorporados en la parte del cálculo.

En este capítulo se describe la aproximación programática utilizada para el cálculo de la correlación entre matrices.

3.1 ELEMENTOS UTILIZADOS

Los elementos que intervienen son, por este orden:

3.1.1 La matriz origen

Compuesta por unos y ceros. Se recibe traspuesta, de modo que los elementos a comparar sean los vectores horizontales.

3.1.2 La matriz auxiliar

Creada mediante un script de Python. Es una matriz formada por pares i,j. Es simétrica y de dimensiones iguales al número de elementos a comparar (típicamente, la dimensión menor de la matriz origen). Su existencia es clave en el proceso de paralelización: el cálculo de la matriz de correlación se realiza para cada par de vectores, y el conjunto de pares de vectores debe ser dividido entre todos los nodos del clúster, para que cada uno realice sólo su parte. Lo que se envía a cada nudo es una parte de esta matriz auxiliar, con el "listado" de pares de vectores que corresponde comparar a ese nodo.

Esta matriz será la recorrida para calcular los coeficientes.

3.1.3 Script de lanzamiento

Es un *script* de *Shell* que realiza la copia de archivos, el lanzamiento del programa Python de cálculo de correlación y, al finalizar éste la recogida de los resultados y su reunión en la matriz solicitada.

3.1.4 Programa de cálculo de correlación

Es el *script* de Python que va recorriendo la matriz auxiliar, calculando la correlación entre cada par de valores.

3.2 PROCESO DE CÁLCULO

El proceso de cálculo de correlación entre líneas de una matriz es relativamente simple: basta ir tomando las líneas de dos en dos, y calcular la correlación entre los dos vectores, formados por todos los elementos de cada línea.

Este recorrido es de fácil programación: bastan dos bucles anidados: uno que recorra la lista de líneas y, dentro de ella, para cada línea ***i***, otro que vuelva a recorrer la lista, realizando, para cada línea ***j***, el cálculo ***corr(i, j)***.

Esta estructura no es paralelizable en entornos Hadoop o Spark. Ambos sistemas se basan en dividir el conjunto de datos en particiones menores, de forma que se entrega cada una de estas particiones a una unidad de cálculo, el *container*. En cada una de estas unidades se ejecuta el mismo programa, que deposita el resultado en un lugar conocido, para que otro proceso lo recoja y reúna los fragmentos resultantes.

El problema consiste en que, en este caso, son necesarias las comparaciones de todas las líneas con todas las líneas. Si se divide la matriz de n líneas en, por ejemplo, cuatro grupos de $\boldsymbol{n / 4}$ líneas cada uno, y se entregan a cuatro *containers*, cada uno de éstos calculará sólo $\boldsymbol{n / 4 * n / 4}$ pares de líneas, $\boldsymbol{n^2 / 16}$. Los cuatro calcularían en conjunto $\boldsymbol{4 * n^2 / 16}$, o lo que es lo mismo, $\boldsymbol{n^2 /4}$. Mientras que el total a calcular es de $\boldsymbol{n*n = n^2}$.

La solución adoptada consiste en crear una matriz auxiliar, formada por parejas ***ij*** de líneas de la matriz. Esa matriz tendría unas dimensiones de $\boldsymbol{n^2}$ **x 1**.

Esta matriz es la que se dividiría entre las unidades de cálculo, los *containers*. Dado un número ***m*** de containers, cada uno de ellos recibiría sólo $\boldsymbol{n^2 / m}$ líneas.

Previamente a esta operación, se habrá copiado, a cada uno de los nodos físicos, la matriz dada completa.

Cada uno de los *containers* ejecutará un sencillo programa Python que tomará, uno por uno, los pares de números ***ij***. A continuación, para cada ***ij***, seleccionará, de la matriz dada, las líneas que correspondan a los dos números. Calculará la correlación entre ambos vectores y proporcionará un número racional correspondiente al valor de la correlación (*float*). Con todos los valores correspondientes a la línea ***i***, formará una lista que guardará como una línea en el archivo resultante.

Una vez finalizado el cálculo para todas las líneas, el archivo resultante será depositado en un directorio común (*output*) en el clúster.

El script principal recoge todos los archivos resultantes y los deposita en la estación local. Una vez en ella, un programa Python reúne los archivos y crea un archivo ***.csv*** con el resultado de la matriz. A continuación, los nodos son destruidos, con lo que deja de generarse gasto.

El investigador dispone, pasados unos minutos, y con un coste que puede ser estimado con antelación, de una matriz con los coeficientes de correlación para cada línea de la matriz original.

3.2.1 Simetría de las matrices

El capítulo anterior ha mostrado, de modo sintético, las funciones realizadas por la herramienta. Es necesario realizar una salvedad sobre la simetría de las matrices. La matriz auxiliar, formada por el cruce de todas las líneas entre sí, es simétrica. Quiere decir que el cálculo realizado de todas con todas es redundante. Y que, para economizar costes de cálculo, debería sólo ser calculado el 50% de los pares de elementos.

Se ha decidido calcular todos por dos motivos:

1. La eliminación de una cierta cantidad de elementos por línea complica el código, al obligar a introducir un juego de contadores para limitar los cálculos.
2. A la hora de descomponer y recomponer la matriz para distribuirla entre los nodos, no es posible enviar elementos parciales con diferentes dimensiones. Sería necesario sustituir los elementos no calculados de la matriz por "0", "N/A", o un valor similar. El hecho de realizar el paso del bucle para la creación de este valor produce un coste computacional prácticamente igual al de calcular el coeficiente de correlación. Por lo que, en aras de la simplicidad y menor coste de programación, se calculan todos los posibles valores de la matriz, independientemente de que sean repetidos.

3.3 DESARROLLO DEL PROCESO

El proceso completo puede dividirse en cuatro fases:

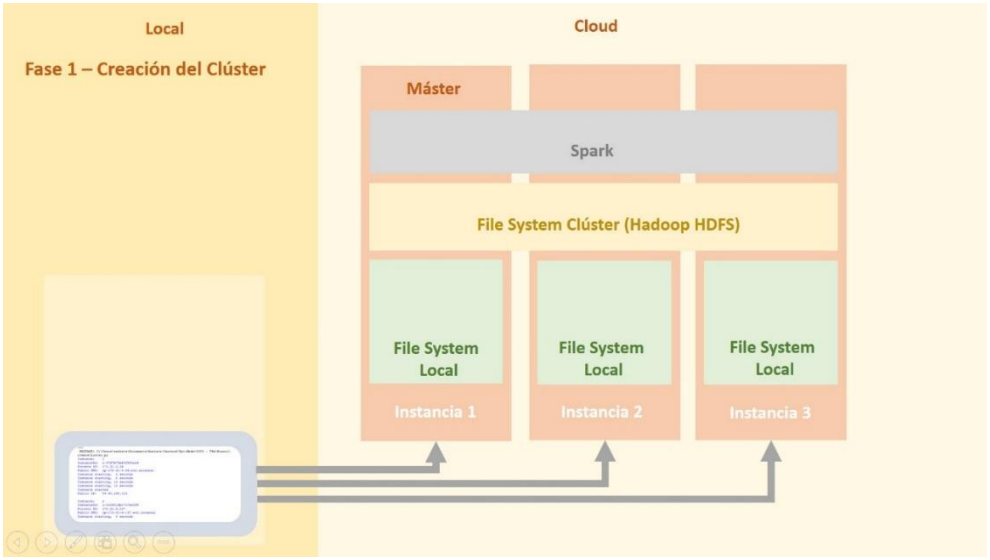
1. Creación del clúster.
2. Preparación del cálculo.
3. Cálculo de correlación.
4. Entrega del resultado.

3.3.1 Creación del clúster

Desde la estación de gestión, el investigador lanza el proceso. Para cada uno de los futuros nodos del clúster, crea una instancia, con Hadoop y Spark ya instalados, a partir de la imagen AMI ya citada.

Cada instancia dispone de un sistema de ficheros propio, junto al que se crea el sistema de ficheros de clúster, compartido entre todos los nodos (HDFS).

También se prepara el sistema Spark, que utilizará HDFS, y que se distribuye por todos los nodos.



Estructura del sistema. Con el área Local, formada por la estación de gestión, y el área de Cloud, formada por el clúster

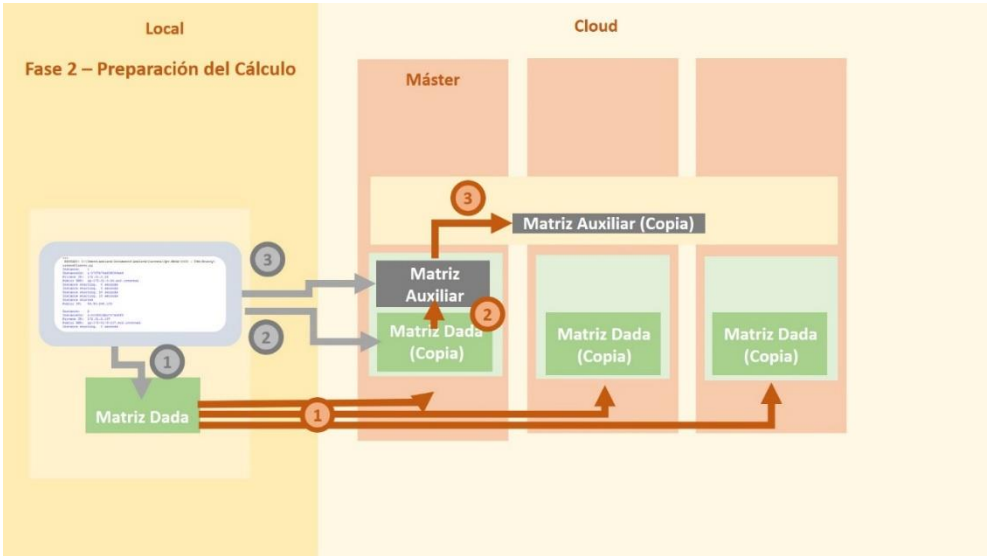
3.3.2 Preparación del cálculo

Desde la estación de gestión, el programa realiza sucesivamente tres operaciones:

1. Se copia la matriz dada a cada uno de los nodos, en el sistema de ficheros propio de la instancia. (La "matriz dada" es la que se ha proporcionado, y para la que se desea obtener las correlaciones). De este modo, cada uno de los contenedores de Spark dispondrá de todos los vectores que su parte de la matriz auxiliar le pida correlacionar.

2. Crea una matriz auxiliar con todas combinaciones posibles de las dimensiones, tomadas de dos en dos. Es una matriz de dimensiones $n^2 \times 1$, compuesta por pares ij. Cada uno de estos valores hace referencia a un registro de la matriz original.

3. Copia la matriz auxiliar creada al sistema de archivos del clúster, en HDFS.



Preparación del cálculo, con creación de la matriz auxiliar y copia al clúster de ésta y la matriz dada

3.3.3 Cálculo de correlación

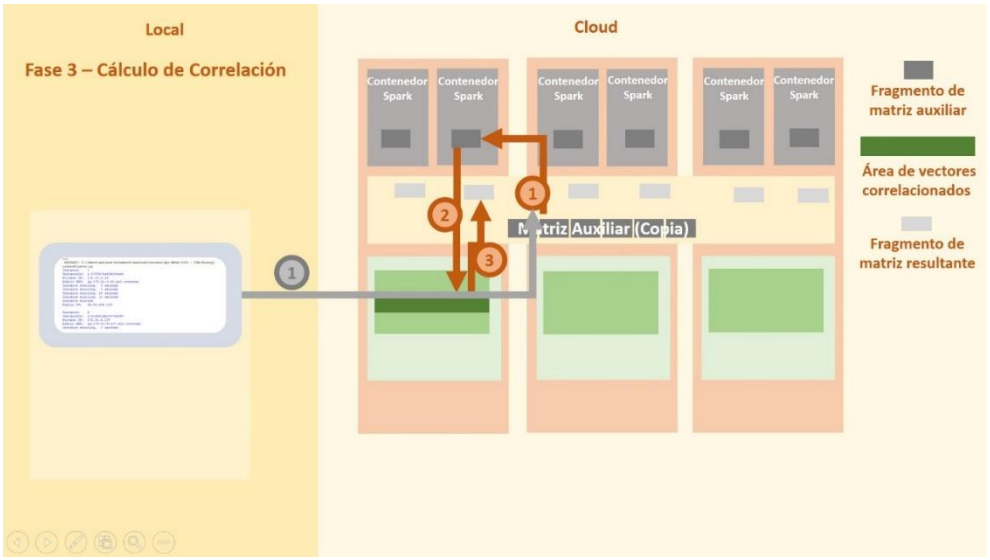
Desde la estación de gestión, el programa continúa, lanzando un script en *pyspark* que se ejecuta en el máster. Este script hace que Spark:

1. Divida la matriz auxiliar en tantas particiones como se le indica, creando un *container* para cada una de estas particiones.

2. En cada uno de los contenedores, y para cada par ***ij***, lea los vectores completos correspondientes a las dimensiones ***i*** y ***j***, de la copia de la matriz dada que se encuentra en el sistema de ficheros de la instancia.

3. Deposite el coeficiente de correlación de cada par en un fichero en Hdfs.

Este proceso se realiza en paralelo para cada uno de los contenedores de Spark, dividiendo la tarea de cálculo entre todos los nodos del clúster, como se pretendía.

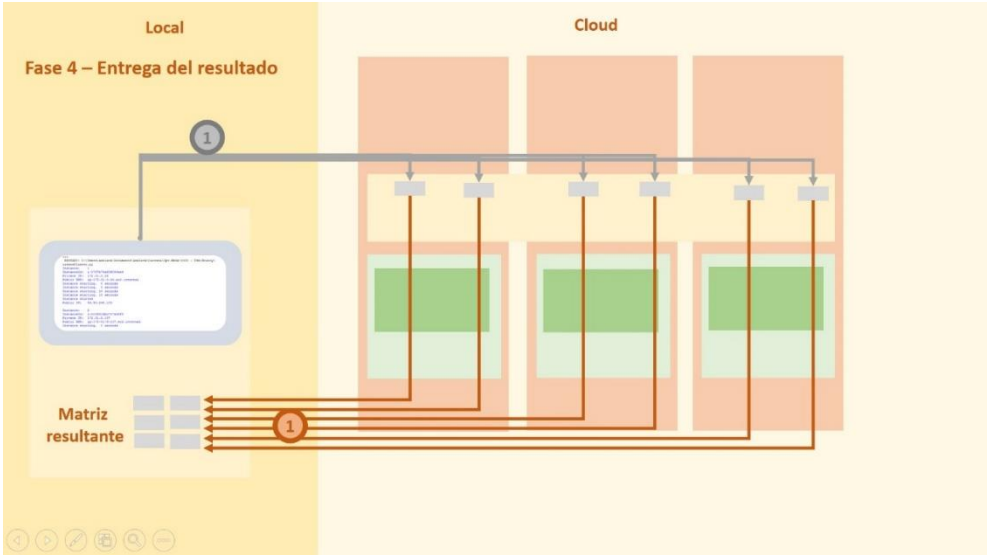


Cálculo de correlación. Cada nodo lee secuencialmente, de la matriz dada, los pares i,j que le corresponden, y realiza la correlación

3.3.4 Entrega del resultado

Una vez todos los contenedores han finalizado sus tareas, se dispone en el clúster, en HDFS, de tantos fragmentos de la matriz resultante como contenedores de Spark ha habido en funcionamiento.

La estación de gestión los recoge, reuniéndolos en el disco local del investigador y formando con ellos la matriz de correlaciones que se pretendía obtener.



Entrega del resultado. Los fragmentos creados por cada nodo son reunidos en una matriz resultado en la estación de gestión

4 CONCLUSIÓN

El trabajo intenta cubrir el espacio existente entre el desarrollo de la tecnología y las necesidades de los investigadores.

La tecnología de clúster en nube y computación en paralelo ofrece mejoras en la capacidad de cálculo, pero introduce necesidad de nuevos conocimientos técnicos.

La herramienta propuesta oculta esa complejidad a los investigadores, permitiéndoles utilizarla directamente para el caso de uso ofrecido, y con muy poco esfuerzo y necesidad de inversión en conocimientos, en otros casos posibles.

Los tres criterios principales de diseño han sido cumplidos:

1. Economía. Se utiliza computación en clúster de pago por uso, restringiendo al máximo, y de modo automático, el coste. No existen tiempos muertos. Sólo existe la posibilidad de un error que corte el proceso y, para ello, se incorpora una utilidad de eliminación manual de las instancias activas.

2. Sencillez. Se ofrece una solución completa que ofrece una funcionalidad habitual entre los investigadores, de modo cerrado y automático. El uso de instancias que se crean y se destruyen en el momento, simplifica los costes de mantenimiento de la seguridad, junto al uso de herramientas sencillas, como Python.

3. Control. Se ofrecen todos los elementos del proceso en modo abierto. Se utilizan bibliotecas estándar y ampliamente difundidas. Se utiliza un lenguaje de programación sencillo y ampliamente difundido.

Con ello, se pone a disposición de la comunidad investigadora una herramienta que le permite aprovechar los avances en sistemas de cálculo para grandes cantidades de datos.