Marcos Esteve Hernandez
Alberto Blanco Álvarez

Práctica 1. Multiplicación de matrices mediante MapReduce

La realización de la práctica ha sido en todo momento conjunta por parte de los dos integrantes del grupo, Alberto Blanco Álvarez y Marcos Esteve Hernández, ya que hemos ido programando las funciones de manera conjunta, recabando información de páginas web y debatiendo las ideas sobre cómo realizar las distintas operaciones. En el video explicativo, hemos intentado dividir la explicación en partes iguales pero queríamos dejar constancia que las explicaciones que hacemos no corresponden con las partes individuales que ha hecho cada uno, sino que ambos hemos trabajado conjuntamente en todas las funciones de la práctica.

Funciones elaboradas

Función iniMatrix(int workers, int n_fil_mA, int n_col_mA, int n_fil_mB, int n_col_mB, ibm_cos) retorna String[]

Esta función recibe los parámetros iniciales especificados en la función main() para generar las matrices de forma aleatoria, basándose en el módulo numpy.

Después, se fijan las filas/columnas que tratará cada worker. Se ha realizado una implementación en la cual, los workers tratan las filas de la matriz A, dividiéndoselas de manera equitativa, y tratan todas las columnas, en el caso de que sea el número de workers igual o menor al número de filas. (El último worker tratará las filas que resten por tratar, si se da el caso de que no sea divisible nºfilas A / workers). También puede ocurrir que haya un worker para cada fila de la matriz A y para cada columna de la matriz B (número de workers = filasA*colB).

Tras haber dividido la matriz según los workers, el resultado de cada submatriz se guardará en diferentes ficheros, que se subirán al servicio de IBM Cloud, al bucket especificado. El fichero tendrá el siguiente formato: Subdivisión de matriz (A o B) + número para identificar + .txt (Ej: SubA1.txt). Finalmente, se construye un vector de Strings que guardará en cada elemento la operación que se ha de realizar para multiplicar la matriz, es decir, los dos ficheros que se deberán obtener del IBM cloud para poder multiplicar la fila y columna respectivas. En cada posición habrá, con el siguiente formato, especificada una operación: NomFicheroA*NomFicheroB, donde NomFicheroA y NomFicheroB siguen los formatos especificados anteriormente.

Función my_map_function(String[] result, ibm_cos) retorna Diccionario

Esta función recibe el string generado en la inicialización de las matrices, el cual especificará la multiplicación de qué dos ficheros ha de realizar el worker.

Para esto, divide el string en los dos fragmentos que corresponde, descarga la submatriz A y la submatriz B, se deserializa el stream de bytes que contiene mediante una función de pickle (pickle.loads), y se realiza la multiplicación de las dos submatrices mediante la función dot.

Una vez hecho esto, se prepara el diccionario. Para ello, utilizaremos como claves para identificar cada valor del diccionario el valor numérico del fichero que contenía la submatriz A, el carácter A, y lo mismo con la B. Es decir, quedará como clave el formato SubmatrizASubmatrizB, con valores numéricos que permitirán ordenarlo en el orden que corresponda para recomponerlo en la función reduce. A esta clave, le asignamos como valor el resultado de la multiplicación pasada como lista (tolist()).

Función my_reduce_function(Diccionario results) retorna Matriz[][]

Para la función reduce utilizamos un diccionario. Primeramente, a partir del resultado del map, se utiliza la función update, la cual funciona de la siguiente manera: se le pasa por parámetro otro diccionario y, si la clave no aparece en el primer diccionario, el elemento (clave: valor) se añade; si la clave coincide con alguna en el diccionario, se actualiza el valor. De esta forma, obtenemos un diccionario con las parejas clave – valor correspondientes a la ejecución de los workers del map.

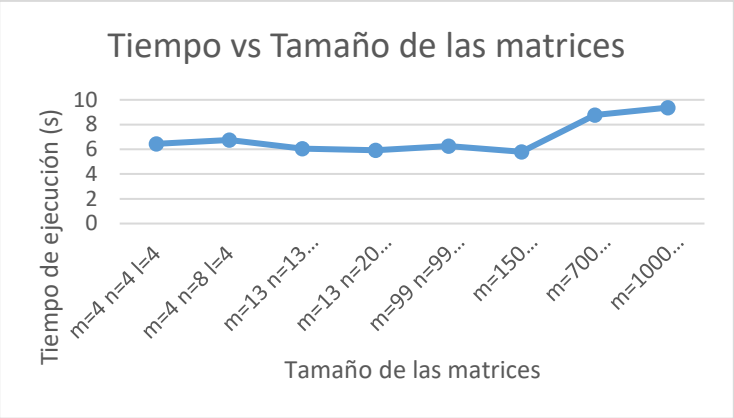
A continuación, iteramos el diccionario y utilizamos la función extend para obtener un vector a partir del diccionario. Esta función concatena una lista o vector a otro.

En este punto, si cada worker había calculado una fila entera o más, la matriz resultante ya queda con el formato adecuado y se retorna como resultado final de la operación. Sin embargo, si el número de workers era igual a la multiplicación de filas de A por columnas de B, es decir, que cada worker habrá calculado un número de la matriz final, se debe operar para que quede una matriz con el formato adecuado.

Para ello, traemos las variables globales workers, n_fil_mA y n_col_mB para conocer el tamaño de la matriz resultado y los workers utilizados, e inicializamos un par de matrices auxiliares. Una de ellas será la encargada de ir montando la matriz fila por fila, mientras que la otra almacenará el resultado final. De esta forma, la matriz queda en este caso también con el resultado final con el formato adecuado y es retornada por la función my_reduce_function.

Tiempo vs tamaño de la matriz

Workers = 1.

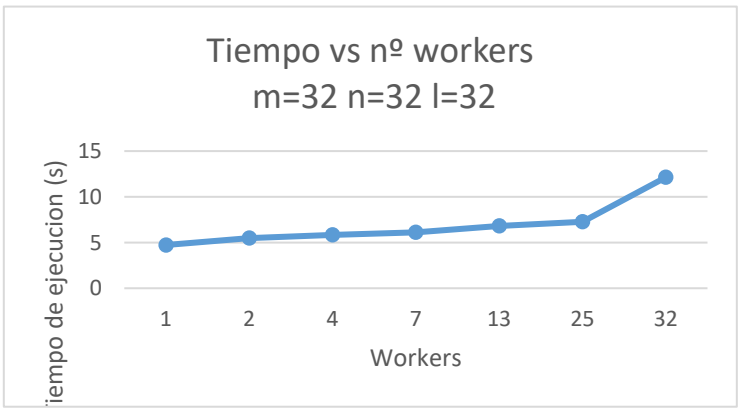


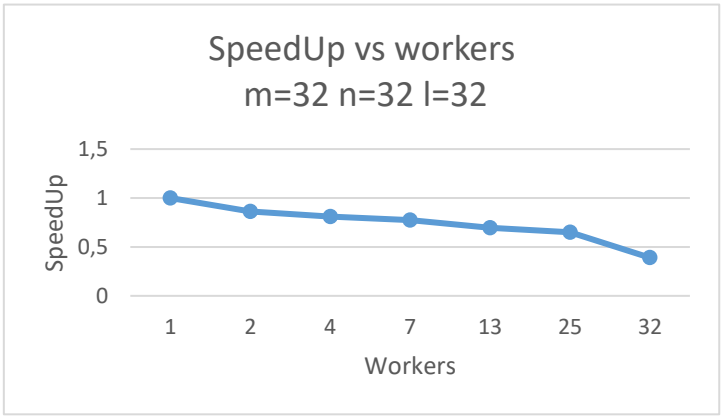
En el gráfico superior tenemos representado en el eje Y el tiempo de ejecución en segundos y en el eje X los distintos valores de 'm', 'n' y 'l'.

Podemos observar que a diferentes tamaños dentro de un rango pequeño, el tiempo de ejecución presenta una tendencia invariable. Esto es debido a que las operaciones de cálculo de la matriz son rápidas y con pequeñas diferencias en el tamaño de esta no se aprecian cambios significativos; la diferencia de tiempo se puede achacar al tiempo de conexión puntual con la red en el momento de ejecutar el programa. Sin embargo, al utilizar tamaños más grandes de la matriz, si que se observa claramente como hay un incremento del tiempo de ejecución. Estos resultados son los esperados ya que a mayor tamaño de matriz, mayor cantidad de operaciones ha de procesar y más tiempo requiere.

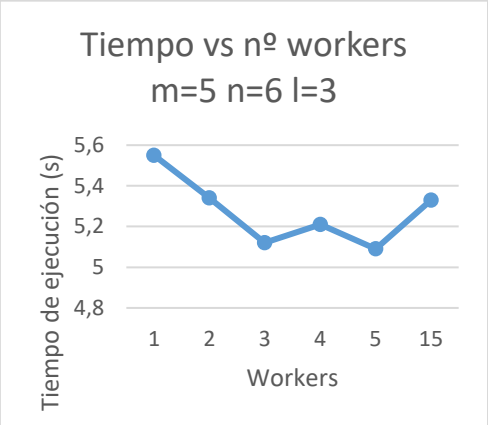
Tiempo vs nº de workers

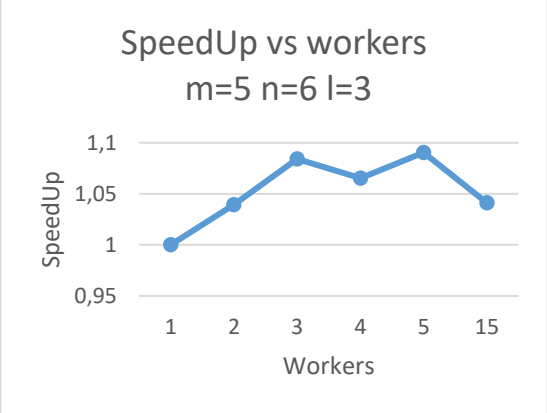
En esta prueba analizamos el efecto del número de workers usados sobre el tiempo de ejecución. Realizamos dos pruebas, fijando el tamaño de las matrices a: dos matrices cuadradas de 'm'='n'='l'=32; y dos matrices irregulares 'm'=5 'n'=6 'l'=3.





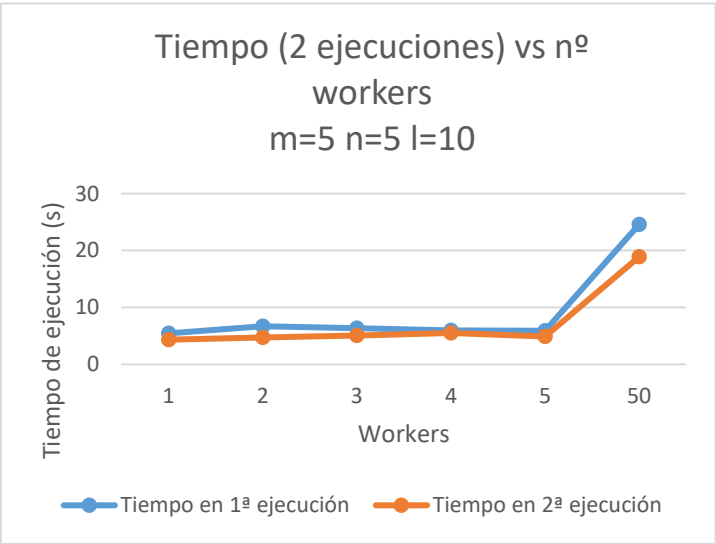
Esta prueba la evaluamos utilizando 'w' = 1, 2, 4, 7, 13, 25 y 32, de modo que la subdivisión en matrices era diferente entre pruebas. Se observa una ligera tendencia creciente en el tiempo hasta 'w'=25, momento a partir del cual el tiempo de ejecución crece de una manera más brusca. Este resultado lo achacamos a que, pese a que cada worker individualmente tiene que calcular menos proporción de la matriz final (una sola fila), los tiempos de creacion de los diversos threads y del filtrado de resultados por parte del reduce son mayores, resultando en total en un tiempo global de ejecución mayor. Como consecuencia, obtenemos que el speedup es menor al aumentar el número de workers, ya que el tiempo de ejecución está aumentando.





En general el tiempo de ejecución disminuye a medida que aumentamos el número de workers. En esta prueba hemos incluido 'w'=filasA*columnasB, ya que no excedía el límite de 100 workers como en el caso anterior. Con 'w'=15, se observa un ligero aumento del tiempo de ejecución. Esto lo achacamos, una vez más, a que el tiempo necesario para lanzar la ejecución 15 veces y el proceso de filtrado de resultados elevan el tiempo de ejecución total, ya que, al tratarse de una matriz pequeña, los tiempos de cálculo de la matriz no son relevantes. En este caso el speedup es mayor al aumentar el número de workers, acorde con la disminución del tiempo de ejecución.

Además nos percatamos que la primera ejecución al fijar los parámetros tardaba más tiempo que la segunda ejecución de manera consecutiva, sin modificar los parámetros. Esto viene reflejado en la siguiente gráfica:



La diferencia de tiempo es ligera, pero se encuentra. Pensamos en el motivo de este hecho, pero no hemos sabido hallar una respuesta en claro. Primero pensamos que era debido a los archivos temporales que se tienen que crear para la ejecución en el cloud, pero los archivos que contienen las matrices se actualizan cada vez, no afecta en eso los parámetros como el número de workers o el tamaño de las submatrices. Otra opción plausible es que la diferencia de tiempo se deba únicamente a la conexión puntual en el momento de ejecutar, pero es significativo que todas las veces que lo ejecutamos, el segundo tiempo fuera menor; de ser problema de la conexión, esto debería pasar siendo la primera o la segunda ejecución, indistintamente. Es posible que este juego de pruebas no sea relevante, pero hemos decidido incluirlo en la documentación porque pensamos que es un dato curioso.

También, se han probado a ejecutar matrices cuyos valores fueran erróneos, o no permitieran la ejecución. Es decir, matrices donde las columnas de A son diferentes a las filas de B. En estos casos, se imprimirá que no fue posible la multiplicación.

También, en el caso de que se introduzca un valor no viable de workers (que sea mayor del número de filas de A, y sea diferente a la multiplicación de filas de A y columnas de B), se dará un mensaje que indica que era erróneo, y se ha cambiado a ejecución secuencial

Prueba de multiplicación:

$$\begin{pmatrix} 2 & 3 & 3 \\ 4 & 7 & 3 \\ 1 & 2 & 6 \end{pmatrix} \times \begin{pmatrix} 4 & 5 & 7 \\ 7 & 8 & 3 \\ 5 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 44 & 37 & 32 \\ 80 & 79 & 58 \\ 48 & 27 & 31 \end{pmatrix}$$

```
ExecutorID a6aa9b/0 - Cleaning temporary data
[[44, 37, 32], [80, 79, 58], [48, 27, 31]]
Tiempo: 7.95 segundos
```

Conclusión

Mediante la segmentación del código, y la ejecución de este en diferentes workers y recopilación posterior de los resultados en una única matriz permite realizar multiplicaciones de matrices de gran tamaño en tiempos significativamente inferiores.

Sin embargo, este método puede resultar contraproducente en matrices de pequeño tamaño, al invertir mas tiempo en la división del código, subida y bajada de las submatrices a la nube, que al cálculo en si.

Otro factor a tener en cuenta es que este tiempo de ejecución también dependerá de la conexión a los servicios de IBM cloud. Por lo tanto, tampoco se recomendaría la utilización del método MapReduce en el caso de que se carezca de una conexión estable.

Documentación

- GitHub - pywren/pywren-ibm-cloud: CloudButton Toolkit implementation for IBM Cloud Functions and IBM Cloud Object Storage. https://github.com/pywren/pywrenibm-cloud.
- Cálculo de matrices con Numpy. http://research.iac.es/sieinvens/python-course/source/numpy.html
- Empleo del módulo Pickle. https://docs.python.org/3/library/pickle.html