INTRODUCCIÓ

La funció principal de l'aplicació és generar dues matrius i multiplicar-les utilitzant un determinat nombre de workers. L'aplicació utilitza un emmagatzematge al núvol per guardar les matrius a multiplicar, que estaran subdividides per files (en el cas de la primera matriu) i per columnes (en el cas de la segona matriu), i la matriu resultant.

FUNCIONAMENT GENERAL DEL CODI

En primer lloc, es demana a l'usuari que introdueixi el nombre de workers a utilitzar i el bucket on es guarden les matrius. Seguidament, es crida a la funció de inicialitzar per generar les matrius. A continuació, es genera el paràmetre interdata amb els *chunks* que han de multiplicar cada *worker*, que es passarà a les funcions de map i map_reduce. Es crida a la funció map o map_reduce, segons el nombre de workers. Finalment, es crida a la funció get_result per obtenir el resultat de la multiplicació.
Programa principal dissenyat per: Joan Mercé López

Variables globals

- `m`: files de la matriu A.
- `n`: columnes de la matriu A (o files de la matriu B).
- `l`: columnes de la matriu B.
- `numWorkers`: nombre de *workers* disponibles per a realitzar les operacions.

FUNCIONS IMPLEMENTADES

INICIALITZAR

La funció d'inicialitzar és l'encarregada de generar les matrius A i B, amb dimensions (m,n) i (n,l), respectivament; partir les matrius segons el nombre de *workers* indicats per l'usuari i, per últim, serialitzar i emmagatzemar els *chunks* de les matrius al *bucket* passat per paràmetre. Per la crida d'aquesta funció s'utilitza la funció call_async.

Funció dissenyada per: Joan Mercé López

Paràmetres

```
def inicialitzar(bucket, ibm_cos)
```

Els paràmetres necessaris per executar la funció són els següents:

- `bucket`: nom del *bucket* on s'emmagatzemen els *chunks* de les matrius.
- `ibm_cos`: paràmetre reservat que permet tindre accés a l'IBM COS des de la funció.

Funcionament

En primer lloc, es generen les matrius A i B de nombres enters aleatoris dins el rang [0, 10) amb dimensions (m,n) i (n,l), respectivament.

Un cop generades les matrius, es comença a dividir la matriu A segons el nombre de *workers*. Si el nombre de *workers* és inferior al nombre de files de la matriu, es divideixen el nombre de files entre el nombre de *workers*, de manera que es divideixi de la manera més equitativa possible; es serialitzen els *chunks* i s'emmagatzemen al bucket del cloud. En cas contrari, s'agafa cada fila, es serialitza i s'emmagatzema al bucket del cloud.

Un cop dividida la matriu A, es divideix la matriu B. Si el nombre de *workers* és inferior al nombre de columnes de la matriu, es divideixen el nombre de columnes entre el nombre de *workers*, de manera que es divideixi de la manera més equitativa possible; es serialitzen els *chunks* i s'emmagatzemen al bucket del cloud. En cas contrari, s'agafa cada columna, es serialitza i s'emmagatzema al bucket del cloud.

Per últim, es retornen les matrius A i B que s'han generat al començament de la funció.

MULTIPLICAR

La funció de multiplicar, és l'encarregada de multiplicar les matriu A i B depenent del nombre de workers que indica l'usuari. Cada worker per separat multiplica els chunks que li toquen. Amb menys workers significa que cadascun d'ells farà més feina.

Funció dissenyada per: Joel Sola López

Paràmetres

```
def multiplicar(files, col, bucket, ibm_cos, id)
```

Els paràmetres necessaris per executar la funció són els següents:

- `files`: nombre de files que ha de multiplicar.
- `col`: nombre de columnes que ha de multiplicar.
- `bucket`: nom del *bucket* on es llegeix les els *chunks* de les matrius A i B i on s'emmagatzema els *chunks* resultat del resultat de la multiplicació de matrius.
- `ibm_cos`: paràmetre reservat que permet tindre accés a l'IBM COS des de la funció.
- `id`: indica el nombre que s'utilitzarà per crear la clau pels objectes que guardarem al núvol.

Funcionament

En primer lloc, definim variables on guardarem el contingut de les matrius A i B i a on emmagatzemarem els *chunks* (desserialitzats) que estem treballant en aquell moment.
Un cop definides les variables, començarem a llegir les matrius del núvol, multiplicar-les i enviar els resultats segons el nombre de *workers*.

Si el nombre de *workers* es 1, agafarem totes les dades de la matriu A i B del núvol i ho guardarem en una nova variable on guardem els chunks serialitzats. A continuació, desserialitzem els chunks de A i B, els posem directament a les variables de la matriu A i B respectivament i realitzem la multiplicació de les matrius. Un cop realitzada la multiplicació, serialitzem la solució, la guardem en una variable separada i la enviem al núvol. Per últim, si l'usuari ha indicat que només hi ha un worker, es retorna la variable que conté la solució de la multiplicació realitzada no serialitzada.

Si el nombre de *workers* és diferent de 1 i és més petit o igual que el nombre de files de la matriu A o més petit o igual que el nombre de columnes de la matriu B anirem agafant els diferents chunks guardats al núvol segons el nombre de files i columnes que hagi de multiplicar, els desserialitzem i els anirem afegint a la variable on guardarem la matriu A o B. Un cop hem agafat tots els chunks de la matriu A i la matriu B segons el nombre de files i columnes a multiplicar, els hem desserialitzat i els hem guardat en variables per la matriu A i B, realitzem la multiplicació. Un cop realitzada la multiplicació, serialitzem la solució, la guardem en una variable separada i la enviem al núvol. Per últim, si l'usuari ha indicat que hi ha un nombre de workers que compleix les condicions mencionades anteriorment, es retorna la variable que conté la solució de la multiplicació realitzada no serialitzada.

Si el nombre de workers es diferent de 1 i hi ha més workers que nombre de files o columnes a multiplicar llavors fem un bucle tantes vegades com el nombre d'elements que té la taula files, agafem el chunk de les matrius A i B, els desserialitzem, multipliquem els

valors corresponents, afegim els valors de la posició actual de la taula files i la taula col i, junt amb el resultat de la multiplicació, ho afegim a la taula retorn (que hem definit prèviament). Un cop realitzada la multiplicació, serialitzem la solució, la guardem en una variable separada i la enviem al núvol. Per últim, es retorna la variable retorn que conté les files i columnes corresponents junt al resultat de la seva multiplicació per cadascun dels valors.

REDUIR

La funció de reduir és l'encarregada de reconstruir la matriu resultant de la multiplicació de les matrius A i B a partir de les submatrius generades per la funció multiplicar.
Es crida a aquesta funció des de la funció map_reduce.

Funció dissenyada per: Joel Sola López

Paràmetres

```
def reduir(results)
```

Els paràmetres necessaris per executar la funció són els següents:

- `results`: llista que conté totes les submatrius resultants de les multiplicacions efectuades pels *workers*. Segons el nombre de workers, la llista tindrà un format diferent.

Funcionament

En primer lloc, es comprova si el nombre de workers es inferior a m (nombre de files de A) o l (nombre de columnes de B). En aquest cas, s'afegeixen tots els elements de la llista results en una altra llista (matC) i, utilitzant la funció *reshape* de la llibreria *numpy* per obtenir la matriu resultant de la multiplicació de les matrius A i B amb les dimensions adequades.
En cas contrari, es genera una matriu buida (matC) de dimensions (m, l) i s'afegeixen els elements resultant de la multiplicació a la posició correcta de la matriu.
Per últim, es retorna la matriu resultant (matC).

REFERÈNCIES

PyWren Functions and Parameters:

https://github.com/pywren/pywren-ibm-cloud/blob/master/docs/functions.md

NumPy:

https://numpy.org/

Repositori Github

https://github.com/jostrip2/Practica1_SD