



**Universitat Autònoma
de Barcelona**

TextCounter


Document d'Anàlisi, Disseny i Tests

Borja Arcos Comas

Enrique Gómez Becerra

Versió: 1.0


Data: 07/05/2022

 Universitat Autònoma de Barcelona	TextCounter
	Document d'anàlisi, disseny i tests

Full de control de modificacions


Títol	TextCounter, anàlisi i disseny
Versió	1.0
Autor/a	Borja Arcos Comas Enrique Gómez Becerra
Data	07/05/2022

Control de versions		
Versió	Descripció / Motiu	Data
1.0	Document inicial	07/05/2022

 Universitat Autònoma de Barcelona	TextCounter
	Document d'anàlisi, disseny i tests

Índex

1. Anàlisi i disseny del sistema	Pàg. 3
1.1. Objectiu	Pàg. 3
1.2. Contingut.....	Pàg. 3
2. Diagrama de classe.....	Pàg. 4
3. Diagrama de seqüència	Pàg. 4
4. Tests	Pàg. 5
4.1. Proves modificació de nombre de threads	Pàg. 5
4.2. Comparació amb bucle lineal per comptar paraules	Pàg. 6

 Universitat Autònoma de Barcelona	TextCounter
	Document d'anàlisi, disseny i tests

1. Anàlisi i Diseny del Sistema

1.1. Objectiu

En aquest apartat inclourem com hem dissenyat la nostre aplicació per tal que quedi estructurada i legible per tota persona que vulgui entendre el funcionament del codi. També s'inclourà una descripció del sistema i el seu funcionament.

1.2. Contingut:


- **Anàlisi:**

Els requisits inicials eren implementar una funcionalitat map-reduce sense ajuda de llibreries externes que comptés la quantitat de vegades que apareix una lletra en les paraules d'una quantitat n de fitxers, sense comptar les lletres que es repeten dins la mateixa paraula, donant un percentatge de respecte el total de paraules detectades. Aquesta funcionalitat havia de paralelitzar-se amb l'ajut de threads per crear processos diferents alhora. Aquest programa havia de ser guardat en un contenidor docker.

Per aconseguir aconseguir tots els requisits hem utilitzat el llenguatge python en la versió 3, amb l'ajut de la llibreria externa multiprocessing, més concretament pool. Amb aquesta llibreria hem aconseguit crear els processos necessaris per agilitzar l'execució del programa. Per l'apartat del codi python aclarir que diferenciem lletres de caràcters especials o números i passem totes les majúscules a minúscules per comptar totes les lletres per igual, per exemple "C=c". En quant acabi l'execució el programa mostrarà per pantalla els percentatges de les lletres que hi ha als fitxers passats, separats pel nom de cada fitxer.

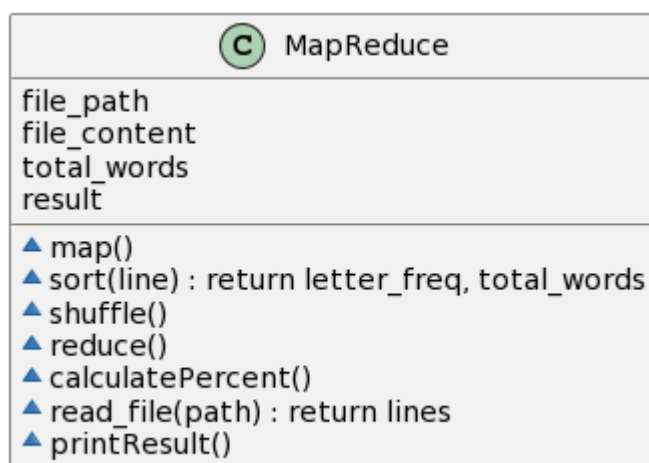
- **Disseny:**

Al ser un codi molt simple, hem utilitzat un disseny single class per fer tota la funcionalitat. Dins d'aquesta classe hi ha 8 mètodes, el constructor entre ells. Cada mètode correspon a un pas del map-reduces excepte 3 que són per mostrar el resultat per pantalla, per calcular el percentatge i per llegir el fitxer. El programa python es crida per mitjà d'un fitxer en bash, que l'usuari serà qui decidirà quan utilitzar-ho. Per altre banda, el contenidor docker es crea a partir d'un dockerfile. Aquest dockerfile conté el necessari per crear el contenidor, descarregar la versió 3 de python i copiar el programa en bash i en python que tenim en local.

 Universitat Autònoma de Barcelona	TextCounter
	Document d'anàlisi, disseny i tests

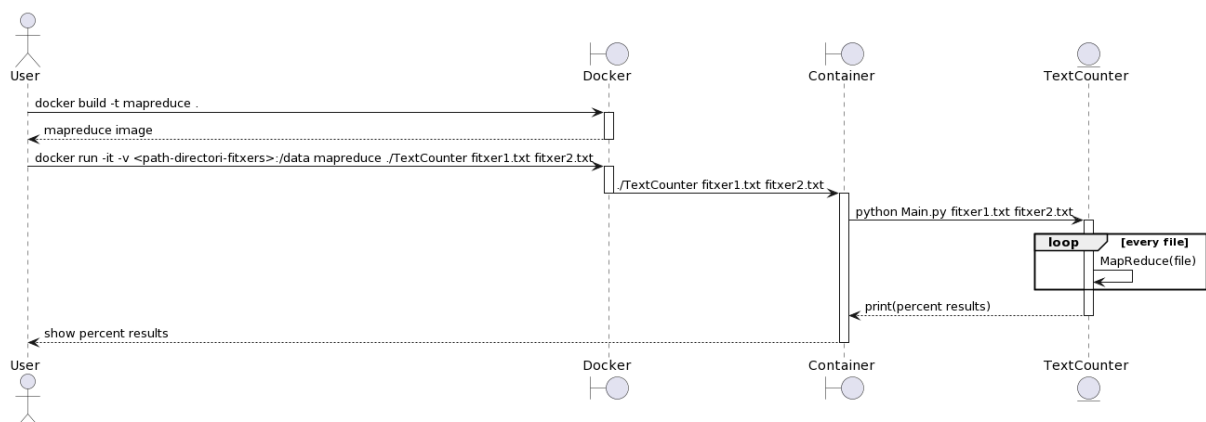
2. Diagrama de classe


Com es podrà veure en el diagrama de classes, només hem hagut d'implementar 1 per solucionar el problema. Dels mètodes utilitzats només 2 retornaran algun valor, ja que la resta utilitzaran el self, treballant així amb variables locals de la classe com file_path, result, etc.



3. Diagrama de seqüència

Al següent diagrama de seqüència s'especifiquen les crides entre els elements pertanyents al sistema: usuari, Docker, container i el propi sistema TextCounter.



 Universitat Autònoma de Barcelona	TextCounter
	Document d'anàlisi, disseny i tests

4. Tests

Hem realitzat una sèrie de proves sobre el sistema modificant el tamany de les dades d'entrada i el nombre de threads en que es divideix l'execució del Map-Reduce. A continuació expliquem en que han consistit concretament aquestes proves i els valors emprats.

4.1. Proves modificació de nombre de threads

Hem anat modificant el número de threads per comparar el rendiment del programa i podem veure la funció pool es on es dedica més temps i per tant per tamany petits de fitxer no és òptim utilitzar quantitats grans de processos diferents.

16 threads 200mb

Name	Call Count	Time (ms) ▼
TextCounter.py	1	364133 100,0 %

<method 'acquire' of '_thread.lock' objects>	20	310265
--	----	--------

32 threads 200 mb

Name	Call Count	Time (ms) ▼
TextCounter.py	1	820503 100,0 %

<method 'acquire' of '_thread.lock' objects>	20	750073 91,4 %
--	----	---------------


10 threads 300mb

Name	Call Count	Time (ms) ▼
TextCounter.py	1	344303 100,0 %

<method 'acquire' of '_thread.lock' objects>	20	283277 82,3 %
--	----	---------------

8 threads 300 mb

TextCounter.py	1	726335 100,0 %
----------------	---	----------------

 Universitat Autònoma de Barcelona	TextCounter
	Document d'anàlisi, disseny i tests

4.2. Comparació amb bucle lineal

El nostre codi crea threads per repartir el treball de processament de dades en diversos processos i així agilitzar i accelerar el seu funcionament. Les dades d'execució obtingudes amb 2 fitxers d'entrada són les següents:

Name	Call Count	Time (ms) ▾
Main.py	1	42150 100,0 %

A continuació modifiquem el codi font per processar les dades en un bucle lineal, sense crear threads.

```
def map(self):
    """p = Pool()
    result = p.map(self.sort, self.file_content)
    for x, y in result:
        self.result.append(x)
        self.total_words += y"""
    for line in self.file_content:
        freq, num_words = self.sort(line)
        self.result.append(freq)
        self.total_words += num_words
```

Si executem amb els mateixos 2 fitxers d'entrada, ara obtenim les dades d'execució següents:

Name	Call Count	Time (ms) ▾
Main.py	1	64770 100,0 %

Si comparem ambdós resultats, observem que el temps amb un bucle lineal respecte l'operació amb threads passa de 42,1 segons a 64,7 segons. Aquesta diferència de 22 segons deixa veure com operar amb threads accelera el funcionament del TextCounter.