UNIVERSITAT ROVIRA I VIRGILI

Communication models and Middleware

Sistemes distribuïts

Alumnes: Jeroni Molina Mellado

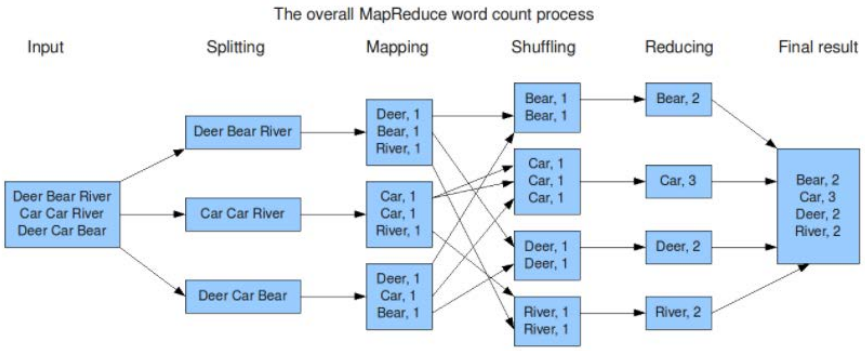
Roger Bosch Mateo

Curs: 2017/2018

Índex

1. Introducció

En aquesta primera practica de sistemes distribuïts, es demana la implementació del model MapReduce utilitzant la llibreria de python pyactor, per a un enunciat específic. El model MapReduce es un model en el qual es crea una paral·lelització de un cert treball, per a que ho realitzin diverses ordinadors alhora per a poder realitzar-lo mes ràpidament. En la següent imatge es poden apreciar tots els passos que implica el model MapReduce:



Tot i això en aquesta practica només es demana implementar la funció Map, que agafarà el input del programa principal i realitzarà la execució del treball, i la funció Reduce que agafarà el output dels diversos mappers, els ajuntarà i mostrarà el resultat final.

L'enunciat de la practica serà implementar un comptador de paraules totals, i un comptador de cada paraula d'un arxiu de text *.txt* , amb seqüencial i implementant el model MapReduce simplificat (mappers i reduce).

2. Implementació

Per implementar el model MapReduce en python utilitzant el pyactor s'ha pensant en utilitzar la opció que permet passar una classe per referencia a qualsevol ordinador extern. Així doncs, la idea es que els ordinadors en els que volem que s'executin els diversos mappers o el reduce, tinguin executant un o diversos hosts, que estaran escoltant contínuament, i en conseqüència enviar des del main el que volem que executin aquests ordinadors,

Amb això ja tenim solucionat el tema de com passar les funcions que es vol que realitzi cada ordinador, però queda solucionar com es farà per a que cada mapper al finalitzar pugui comunicar-se amb el reduce. Per a poder realitzar lo explicat anteriorment, es va decidir per primerament crear la instancia del reduce, i a continuació al cridar al mapper passar-li la instancia reduce per referencia per a que aquest la pugui cridar quan hagi acabat de realitzar la seva feina. Així cada mapper cridarà la funció reduce al finalitzar, i la classe reduce cada vegada que sigui cridarà, recopilarà totes les dades enviades per els diversos mappers, i quan detecti que no hi ha més mappers a processar, mostrarà per pantalla el resultat final.

Ara que ja tenim la estructura del MapReduce, lo explicat en el paràgraf previ s'ha de aplicar en les funcions que hem de realitzar, el comptador de paraules totals, i el comptador de paraules. Primer de tot, es necessita una funció que donat un arxiu de text, el subdivideixi en N parts, per a que cada mapper s'encarregui d'una part del arxiu. Per a realitzar això s'ha creat un script anomenat *splitFile.sh* , que divideix l'arxiu en N parts iguals, creant així diversos arxius anomenats X.part on X serà un nombre del 0 a (N-1). S'ha decidit que la part de subdividir el arxiu no es compti com a temps d'execució total del model MapReduce, ja que el que ens interessa son les funcions Map i Reduce.

Per a que tots els ordinadors puguin accedir als arxius creats per aquesta funció s'ha creat un servidor, que permet que es pugui accedir als continguts del directori on s'executa aquest servidor. La comanda per a crear el servidor es la següent:

```
python -m SimpleHTTPServer
```

Llavors cada mapper el que farà es accedir al contingut de la part del fitxer que li toca llegir, i realitzarà la funció que li diguem. Que pot ser countingWords (comptador de paraules totals) o be wordCounting (comptador de cada paraula). En el cas del countingWords, el mapper contarà el nombre total de paraules del arxiu que li toca llegir. Una vegada finalitzat, cridarà al reduce, passant-li el número total de paraules a aquell arxiu. El reduce, que ja sabrà que estem executant en el mode countingWords perquè li haurem dit amb anterioritat juntament amb el numero total de mappers que ha de processar, rebrà el resultat dels diversos mappers, sumarà tots els resultats i mostrarà per pantalla el nombre total de paraules del arxiu complert, juntament amb el temps d'execució.

En el cas del wordCounting, la base segueix sent la mateixa però utilitzant diccionaris. Cada mapper crea un diccionari, on afegeix les paraules que vagi trobant, i en cada paraula el nombre de vegades que s'ha trobat en la part del arxiu. En acabar de processar la part de l'arxiu, cridar al Reduce passant-li el diccionari per referencia. La classe Reduce que te un diccionari global, agafa el diccionari donat per la classe Map, i agafa tota la informació per afegir-la al seu diccionari global. Una vegada finalitzat, imprimeix per pantalla cada paraula trobada, juntament amb el nombre d'aparicions, i finalment el temps d'execució.

Amb tot això, ja es tindria feta la implementació del que es demana en l'enunciat de la practica. A continuació en l'apart de disseny s'explicaran detalls més concrets de cada un dels arxius que composen la practica.

3. Disseny

3.1. Nmappers

En aquest apartat es parlarà de tots els arxius anomenats Nmappers com 2mappers, 3mappers, 4mappers, etc. Ja que són el mateix arxiu amb les mateixes funcionalitats, però només canviant el nombre de mappers. Aquest arxiu es podria considerar el main del programa, ja que es el que ho inicia tot. Rep per paràmetre el mode que es vol executar, ja sigui CW (countingWords) o WC (wordCounting), i inicia el reducer juntament amb els N mappers seguint els paràmetres especificats a l'arxiu (que s'hauran de modificar cada vegada que vulguis canviar d'ordinadors (IP,port)). Una vegada instanciats, s'inicia el Reducer, passant-li el nombre de mappers, i s'inicien els mappers, passant-li la ubicació del servidor on es troba l'arxiu que ha de llegir i el Reducer com a referencia.

```
'''
@author: Jeroni Molina Mellado
@author: Roger Bosch Mateo
'''
import sys, time
import urllib2, re
import os.path
sys.path.append(os.path.dirname(os.path.dirname(__file__)))
from pyactor.context import set_context, create_host, Host, sleep,
shutdown
from pyactor.exceptions import TimeoutError
from implementation.reduce import Reduce
from implementation.map import Map


if __name__ == "__main__":
    set_context()

    #Validate the entry argument length
    numberOfArguments = len(sys.argv)
    if (numberOfArguments != 2):
        print "python client.py <mode>"
        exit(-1)

    #Check if mode is correct
    mode = sys.argv[1]
```

```
    if ((mode != "CW") and (mode != "WC")):
        print "ERROR: Mode has to be CW (counting words) or WC (words
count)"
        exit(-1)

    IP_COMPUTER1="http://127.0.0.1"
    IP_COMPUTER2="http://127.0.0.1"

    host = create_host(IP_COMPUTER1+':1679')

    #Spawn the reducer
    reducerHost = host.lookup_url(IP_COMPUTER1 +':' + str(1277) + '/',
Host)
    reducer = reducerHost.spawn(1277, Reduce)
    reducer.setNumberOfMappers(2)

    remoteHost = host.lookup_url(IP_COMPUTER1 +':' + str(1278) + '/',
Host)
    host1 = remoteHost.spawn(1278, Map)

    remoteHost = host.lookup_url(IP_COMPUTER2 +':' + str(1279) + '/',
Host)
    host2 = remoteHost.spawn(1279, Map)

    # 8000 is the default port for the HTTP server
    host1.map(mode, IP_COMPUTER1+':8000/' + str(0) + ".part", reducer)
    host2.map(mode, IP_COMPUTER1+':8000/' + str(1) + ".part", reducer)

    shutdown(), IP_COMPUTER1+':8000/' + str(1) + ".part", reducer)

    shutdown()
```

3.2. Host

S'executarà a cada ordinador on es vulgui executar un mapper o un reducer. La seva funció es escoltar contínuament, esperant a que algú li digui el que haurà de fer.

```
'''
@author: Jeroni Molina Mellado
@author: Roger Bosch Mateo
'''
```

```
from pyactor.context import set_context, create_host, serve_forever
import sys

if __name__ == "__main__":

    numberOfArguments = len(sys.argv)
    set_context()

    if (numberOfArguments == 2):
        host = create_host('http://127.0.0.1:'+sys.argv[1]+'/')
        print 'host listening at port '+sys.argv[1]

    serve_forever()
```

3.3. Map

La classe Map serà passada per referencia per a que s'executi en un host indicat per el main. En aquesta classe tenim primerament la funció *map*, que rep per paràmetre la funció que ha de realitzar (CW, WC), la adreça d'on es troba el arxiu que ha de processar en el servidor, i una instancia del *Reduce*. Depenen de la funció que rep map, s'executarà o be *countingWords* o *wordCounting*.

Aquestes funcions llegiran el fitxer i el processaran, i seguidament mitjançant l'instancia rebuda en la funció *map*, cridaran al reduce per passa-li els resultats obtinguts.

```
'''
@author: Jeroni Molina Mellado
@author: Roger Bosch Mateo
'''
import sys, time
import urllib2, re
import os.path
sys.path.append(os.path.dirname(os.path.dirname(__file__)))
from pyactor.context import set_context, create_host, Host, sleep,
shutdown
from pyactor.exceptions import TimeoutError

class Map(object):
    #Asynchronous
```

```
    _tell = ['map']
    _ref = ['map']
    _ask = ['getCW', 'getWC']

    #Reducer actor reference
    reducer = 0
    #Word count list
    wordDic = dict()
    #Count
    count = 0

    # Main function that allows to map a file by counting the number
of words(CW)
    # or by counting each word appereance.
    # Usage:
    #   functionToCall: "CW" or "WC"
    #   httpAddress:    Address for the file
    #   reducer:        reducer actor reference
    def map(self, functionToCall, httpAddress, reducer):
        self.reducer = reducer
        if (functionToCall == 'CW'):
            self.count = 0
            self.countingWords(httpAddress)
        elif (functionToCall == 'WC'):
            self.wordDic = dict()
            self.wordCounting(httpAddress)

    # Function that counts the number of words in a files
    def countingWords(self, address):
        contents = urllib2.urlopen(address).read()
        self.count = len(re.findall(r'\w+', contents))
        self.reducer.reduceCW(self.count)

    # Function that states the number of times a word appears in a
file
    def wordCounting(self, address):
        #Get the file to read
        contents = urllib2.urlopen(address).readlines()
        for line in contents:
            line = line.decode('utf_8')
            words = re.compile(r"[a-zA-Z]+").findall(line)
```

```
            for word in words:
                word=word.lower()
                #If word exists
                if (self.wordDic.get(word)):
                    self.wordDic[word] = self.wordDic[word] + 1
                #If it doesn't exist
                else:
                    self.wordDic[word] = 1

        self.reducer.reduceWC(self.wordDic)

    def getCW(self):
        return self.count

    def getWC(self):
        return self.wordDic
```

3.4. Reduce

La classe Reduce serà passada per referencia en només un ordinador per a que s'executi en un host indicat per el main. Primerament té la funció *setNumberOfMappers*, que es la que s'ha d'iniciar al principi, ja que rep el número de mappers que haurà de processar per a saber quan imprimir el resultat final i finalitzar. Aquesta funció serà cridada per el main.

Després tenim les funcions que *reduceCW* i *reduceWC*, que recopilaran les dades dels mappers, imprimiran per pantalla el resultat final i finalitzaran. Aquestes funcions seran cridades per els mappers al finalitzar.

```
'''
@author: Jeroni Molina Mellado
@author: Roger Bosch Mateo
'''
import sys, time
import urllib2, re
import os.path
sys.path.append(os.path.dirname(os.path.dirname(__file__)))
from pyactor.context import set_context, create_host, Host, sleep,
shutdown
```

```
from pyactor.exceptions import TimeoutError

class Reduce(object):
    #Asynchronous
    _tell = ['reduceCW', 'reduceWC', 'setNumberOfMappers']
    _ask = ['getNumberOfMappers', 'getCW', 'getWC']



    #Number of mappers finished
    nMappers=0
    #Number of mappers to expect
    totalMappers=0

    #StartTime
    start_time=0

    #Total of words for CW
    total=0
    #List of words for WC
    wordCounting = dict()

    def reduceCW(self, count):
        self.nMappers = self.nMappers + 1
        if ( self.nMappers < self.totalMappers):
            self.total= self.total + count
        elif (self.nMappers == self.totalMappers):
            self.total = self.total + count
            print("The number of chracters is "+str(self.total)+"\n")
            #print execution time
            print("Execution time: %s seconds" % (time.time() -
self.start_time))

    def reduceWC(self, wordDic):
        self.nMappers = self.nMappers + 1
        if ( self.nMappers < self.totalMappers):
            for word,count in wordDic.items():
                #If word exists
                if (self.wordCounting.get(word)):
                    self.wordCounting[word] = self.wordCounting[word]
+ count
                #If it doesn't exist
```

```
                else:
                    self.wordCounting[word] = count

        elif (self.nMappers == self.totalMappers):
            for word,count in wordDic.items():
                #If word exists
                if (self.wordCounting.get(word)):
                    self.wordCounting[word] = self.wordCounting[word]
+ count
                #If it doesn't exist
                else:
                    self.wordCounting[word] = count

            finish_time=time.time()
            for word,count in self.wordCounting.items():
                print (word+": "+str(count))

            #print execution time
            print("Execution time: %s seconds" % (finish_time -
self.start_time))



    #This function must be called before starting mapping with the
number of mappers
    def setNumberOfMappers(self, totalMappers):
        self.wordCounting = dict()
        self.total=0
        self.nMappers=0
        self.totalMappers=totalMappers
        self.start_time=time.time()

    def getNumberOfMappers(self):
        return self.totalMappers

    def getCW(self):
        return self.total

    def getWC(self):
        return self.wordCounting
```

3.5. MapSequential

Es la mateixa versió de la classe Map, però implementat per a que en comptes d'utilitzar-se a un servidor, es llegeixi en un mateix ordinador.

```
'''
@author: Jeroni Molina Mellado
@author: Roger Bosch Mateo
'''
import sys, time
import urllib2, re
import os.path

class Map(object):
    #Asynchronous

    #Reducer actor reference
    reducer = 0
    #Word count list
    wordDic = dict()

    def map(self, functionToCall, fileToRead, reducer):
        file = open(fileToRead, "r")
        self.reducer = reducer
        if (functionToCall == 'CW'):
            self.countingWords(file.read())
        elif (functionToCall == 'WC'):
            self.wordCounting(file.read())

    # Function that counts the number of words in a files
    def countingWords(self, inputData):
        count = len(re.findall(r'\w+', inputData))
        self.reducer.reduceCW(count)

    # Function that states the number of times a word appears in a
file
    def wordCounting(self, inputData):
        #Get the file to read
        inputData = inputData.decode('utf_8')
        words = re.compile(r"[a-zA-Z]+").findall(inputData)

        for word in words:
```

```
            word=word.lower()
            #If word exists
            if (self.wordDic.get(word)):
                self.wordDic[word] = self.wordDic[word] + 1
            #If it doesn't exist
            else:
                self.wordDic[word] = 1
        self.reducer.reduceWC(self.wordDic)

    def getCW(self):
        return self.count
```

3.6. ReduceSequential

Lo mateix que en el cas anterior, però amb la classe Reduce.

```
'''
@author: Jeroni Molina Mellado
@author: Roger Bosch Mateo
'''
import sys,time
import urllib2, re
import os.path

class Reduce(object):
    #StartTime
    start_time=0

    #Total of words for CW
    total=0
    #List of words for WC
    wordCounting = dict()

    def reduceCW(self, count):
        self.total = self.total + count
        print("The number of chracters is "+str(self.total)+"\n")
        #print execution time
        print("Execution time: %s seconds" % (time.time() -
self.start_time))
```

```
    def reduceWC(self, wordDic):
        for word,count in wordDic.items():
            #If word exists
            if (self.wordCounting.get(word)):
                self.wordCounting[word] = self.wordCounting[word] +
count
            #If it doesn't exist
            else:
                self.wordCounting[word] = count

        finish_time=time.time()

        for word,count in self.wordCounting.items():
            print (word+": "+str(count))
        #print execution time
        print("Execution time: %s seconds" % (finish_time -
self.start_time))



    #This function must be called before starting mapping with the
number of mappers
    def start(self):
        self.total=0
        self.start_time=time.time()

    def getCW(self):
        return self.total

    def getWC(self):
        return self.wordCounting
```

4. Speedups

Per a realitzar els diferents speedups s'ha utilitzat primerament l'ordinador 1, per a realitzar les proves seqüencials, i seguidament els dos ordinadors per a realitzar les proves amb mappers i reducer. Les característiques dels dos ordinadors són les següents:

	Ordinador 1	Ordinador 2
Tipus	De sobretaula	Portàtil
Sistema Operatiu	Ubuntu instal·lat	Ubuntu corrent a USB
Freqüència processador	Intel Core 2QuadCPU q9300 @2,40 GHz	i5-6200U @2,40GHz
Nº nuclis	4	4

És important recalcar que les proves en distribuït s'han realitzat connectats a internet per cable, ja que es va intentar fer per wifi on els resultats no eren visibles degut al cost de comunicació extra. Inicialment les proves també es van realitzar sobre l'Ubuntu que deix instal·lar Windows però la combinació de wifi i la poca paral·lelització que l'Ubuntu per a Windows permet ens vam veure obligats a instal·lar Ubuntu per a veure resultats.

Els temps de les proves son obtinguts mitjançant la comanda *time()* de python. Al iniciar el programa s'emmagatzema el temps actual, i quan el reduce estigui preparat per a imprimir per pantalla el resultat final, en aquell moment s'emmagatzema un altre cop el temps. Al imprimir per pantalla, el temps final es resta amb l'inicia, obtenint així el temps total d'execució. Es va decidir treure de la quantificació de temps el temps que es tarda en imprimir per pantalla el resultat final, ja que per exemple en el cas del wordCounting es tarda uns segons en imprimir-ho tot i creiem que no es important a la hora de realitzar els speedups.

Els fitxers que s'han utilitzat per a realitzar el joc de proves son els següents:

	Bíblia	Quijote	Sherlock	BigSherlock	GiantSherlock
Mida (M)	1,17	2,16	0,54	69,23	128,46

Els fitxers bigSherlock i giantSherlock, son el llibre Sherlock multiplicat un cert numero de vegades. Es van crear per a poder veure com afectava l'arquitectura MapReduce amb fitxers més grans.

La estructura amb els mappers durant les proves ha sigut la següent:

Nº mappers	1 (Seqüen.)	2	3	4	5	6	7
PC 1	1	1	2	2	3	3	3
PC 2	-	1	1	2	2	3	4

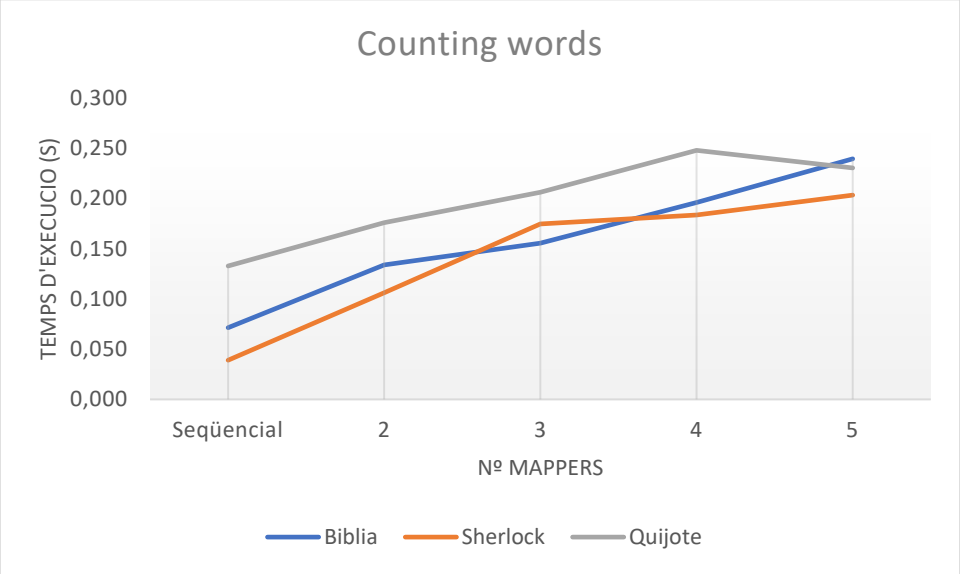
El PC 1 a més dels mappers, ha executat el reducer en tots els casos.

4.1. Counting Words

Dades obtingudes:

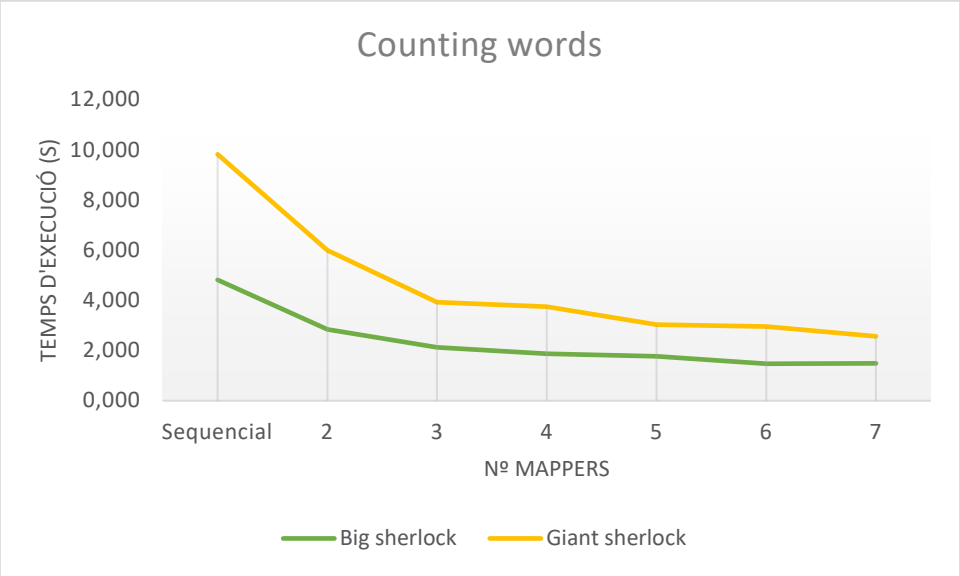
Nªmappers	Seqüencial	2	3	4	5	6	7
Biblia	0,071	0,134	0,156	0,196	0,240		
Sherlock	0,039	0,106	0,175	0,184	0,204		
Quijote	0,133	0,176	0,206	0,248	0,231		
Big sherlock	4,817	2,837	2,133	1,869	1,767	1,474	1,485
Giant sherlock	9,824	5,994	3,922	3,754	3,035	2,962	2,568

A partir dels llibres més petits es pot obtenir el següent gràfic:



En aquest gràfic es pot observar com en la funció CountingWords no surt rentable executar el model MapReduce en arxius de mida petita, ja que no s'arriba a superar mai l'ús seqüencial. Però si que es pot observar que en el cas del Quijote, el llibre de més mida en aquesta gràfica, en el pas dels 4 als 5 mappers, sofreix una millora en el temps. Deixant així entreveure la conclusió en la que s'arribarà mes tard.

Ara es passarà a analitzar el efecte del MapReduce en els llibres més grans:



En aquest gràfic si que es pot observar una gran millora de la execució seqüencial a la execució amb dos ordinadors utilitzant el model MapReduce. A més contra mes mappers s'han utilitzat, el temps ha anat disminuint degut als numero de nuclis dels processadors que permeten una paral·lelització dels processos en cada un dels dos ordinador.

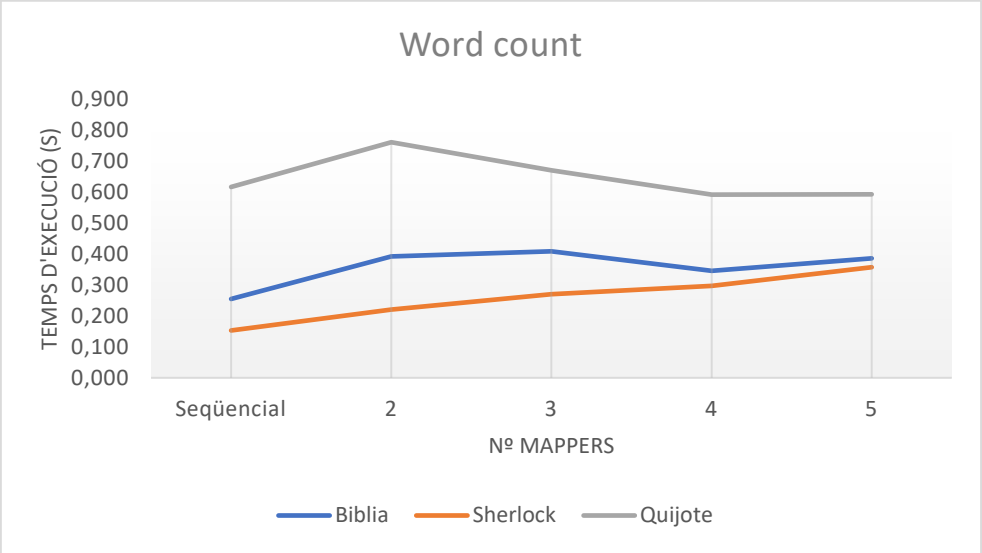
Gracies a aquest gràfic es por arribar a la conclusió que la funció de countingWords funciona millor contra més gran es l'arxiu, degut a que amb un arxiu petit no val la pena paral·lelitzar amb mappers ja que es tarda més connectant amb els hosts, connectant amb el servidor, enviant al reduce, etc, que fent-ho seqüencialment.

4.2. Word Count

Dades obtingudes:

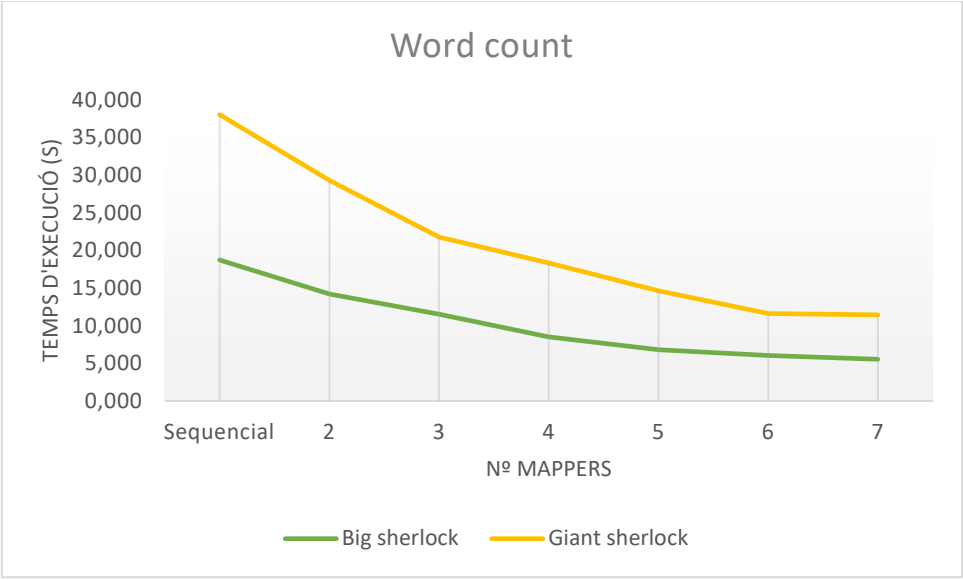
Nªmappers	Seqüencial	2	3	4	5	6	7
Biblia	0,255	0,393	0,409	0,346	0,387		
Sherlock	0,153	0,221	0,271	0,298	0,358		
Quijote	0,617	0,762	0,671	0,593	0,594		
Big sherlock	18,733	14,213	11,540	8,519	6,831	6,053	5,541
Giant sherlock	38,046	29,308	21,764	18,357	14,658	11,625	11,442

A partir dels llibres més petits es pot obtenir el següent gràfic:



S'observa que com en el cas anterior, no val la pena en arxius petits. Però tot i així la mida del arxiu en la qual comença a ser rentable es més petita que en la funció countingWords com s'observa en el llibre Quijote, que arriba a igualar la execució seqüencial.

I a partir dels llibres més grans s'obté la següent:



Observant la gràfica es pot arribar a una conclusió similar a l'apartat anterior, veient que en arxius grans o molt grans surt plenament rentable utilitzar el model MapReduce.

4.3. Conclusions speedup

- *Counting words* és una funció molt mes ràpida que *word counting*, és a dir *word counting* té un cost major.
- Els resultats es comencen a veure més ràpidament per a *word counting* que per a *counting words*.
- Per a fitxers petits no surt a compte treballar en distribuït. El cost de comunicació entre els ordinadors amb el *middleware PyActor* és massa gran.
- Per a fitxers grans surt a compte treballar en distribuït. El cost de comunicació és assumible respecte al temps de càlcul i s'aconsegueixen millores d'execució del 25% per a *counting words* i del 30% per a *word counting*.
- La paral·lelització arribarà un punt en el que per la mida dels subfitxers ja no ens sortirà a compte. Per exemple del 6è al 7è mapper la diferència de temps és més menor.