PRÁCTICA 1

Communication models and middleware

URV 2018

Sistemas distribuidos

Manuel Ruiz Botella

Cristina Izquierdo Lozano

Índice

Arquitectura y validación de la solución 2
server.py 2
mapper.py 2
reducer.py 2
secuencial.py 3
script_ini.py 3
Código 4
server.py 4
mapper.py 6
reducer.py 7
secuencial.py 9
script_ini.py 12
Juego de pruebas (speedup) 13
Conclusión 15

Arquitectura y validación de la solución

Decisiones de diseño y estructura del programa.

server.py

Es el programa principal que controla el proceso.

Recoge los parámetros para pasarlos a la función de readFile, posteriormente crea un host y spawnea el server en este, después recoge los diferentes host de mapa para cada uno de los maps a partir de su url, y hace lo mismo con el reducer, después spawnea el reducer y realiza la función de readFile.

Dentro de readFile.

Se encarga de leer el fichero y dividirlo según el número de slaves definidos por parámetro.

Éste guarda las particiones del fichero en la carpeta "/partes" que se encuentra en el servidor web, de manera que después el mapper puede usar la dirección IP para leer la partición correspondiente.

También se encarga de hacer el spawn a cada uno de los maps y ponerlos a realizar la función map.

Además, inicia el contador de tiempo antes de llamar al primer mapper, el cual se para al finalizar el reducer. De esta manera sólo cuenta el tiempo que ha estado en funcionamiento el MapReduce como tal.

mapper.py

Se crean tantos host para mappers como slaves y cada uno de ellos se asigna a la ip_local cambiando el puerto para cada host.

Cada uno de los mappers usará una de las particiones del texto a trabajar.

Primero trata el texto que encuentra: pasando todo el texto a minúsculas, eliminando símbolos y signos de puntuación. Una vez limpio, crea un diccionario con las palabras (clave) y las veces que aparecen en esta porción del texto inicial (valor).

Finalmente envía al reducer esta parte del diccionario para que lo junte con los resultados del resto de mappers.

reducer.py

Solo hay un reducer que se crea y sirve continuamente.

Recibe el resultado de cada uno de los mappers y suma los valores de las claves (palabras) uniéndolos en un nuevo diccionario. Posteriormente, según si el usuario ha escogido usar el CountingWords o el WordCount, éste dará una salida u otra y parará el tiempo al finalizar.

En cualquier caso, la función usada es la misma. Para ahorrar código, hemos decidido que el tipo de programa será un boleano y que una vez tenga el diccionario creado y todos los valores sumados, dará un resultado u otro:

- **CountingWords**: suma los valores de cada clave y devuelve el total de palabras.
- **WordCount**: para cada clave, imprime por pantalla ésta y su valor. Es decir, que imprime cada palabra con el número de apariciones en el texto.

En esta misma clase hemos definido las funciones para iniciar y parar el tiempo.

En iniciar tiempo, que se llamaba desde el server, ademas de iniciar la cuenta de tiempo se define el número de slaves que tiene que esperar el reducer y el programa que se tiene que ejecutar (wc o cw).

secuencial.py

Contiene el map y el reduce en un único fichero para poder usar el programa en secuencial.

script_ini.py

Script creado para inicializar las diferentes partes y setear los argumentos pasados por parámetro: nombre del fichero, número de slaves, programa que quiere usarse (WordCount o CountingWords) y las IPs local y del servidor.

Este script abre un terminal para cada uno de los mappers y para el reducer, así nos aseguramos de aprovechar todos los cores, ya que Python por sí solo no lo hace (cada uno sería un thread y no un proceso).

Código

Solución del problema en python.

server.py

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

import commands, sys, io, os, urllib


class Server(object):

      _tell = ['readFile']

      _ask = []

      _ref = ['readFile']


      def readFile(self, hosts_maps, reducer, ip_server, fichero,
      slaves, countingWords):

            urllib.urlretrieve(ip_server+"/"+fichero, fichero)

            fich  = io.open(fichero, "r", encoding="utf-8-sig")
            lineas_total = int(commands.getoutput("wc -l
      "+fichero+" | cut -d ' ' -f1")

            num_lines_por_mapa = lineas_total/slaves

            if (lineas_total % slaves) != 0:

                  num_lines_por_mapa += 1

            maps={}

            for cliente in range(0, slaves):

                  maps[cliente] =
hosts_maps[cliente].spawn('mapa'+str(cliente), 'mapper/Mapper')

            directory = os.getcwd()

            for cliente_d in range(0, slaves):

                  filenueva =
io.open(directory+"/texts/partes/file_"+str(cliente_d)+".txt",
"w", encoding="utf-8-sig")

                  for line in range(0, num_lines_por_mapa):

                        line = fich.readline()

                        if line != "\n":

                              filenueva.write(line)
```

```
                filenueva.close()
          reducer.iniciar_tiempo(slaves, countingWords)
     #iniciamos el tiempo del sistema (entrada al primer mapper)
          for mapa in range(0,slaves):
                maps[mapa].map(ip_server,str(mapa)+".txt",
reducer)
          fich.close()



if __name__ == "__main__":
     set_context()
     try:
          if len(sys.argv) != 6:
                raise IndexError


          fichero = str(sys.argv[1])
          slaves = int(sys.argv[2])
          if (str(sys.argv[3]) == "cw"):
                countingWords = True
          else:
                countingWords = False
          ip_local = str(sys.argv[4])
          ip_server = 'http://'+str(sys.argv[5])+':8000'
     except IndexError:
          print "\nERROR: Argumentos inválidos.\nDeberían
ser:\n1. Nombre fichero\n2. Número slaves\n3. Programa
(CountingWords = cw || WordCount = wc)\n4. IP local\n5. IP
server\n"
     finally:
          host = create_host('http://'+ip_local+':1277')
          server = host.spawn('server', 'server/Server')
          print "server listening at port 1277"
          hosts_maps = {}
```

```
            for num in range(0,slaves):
                  hosts_maps[num] =
host.lookup_url('http://'+ip_local+':130'+str(num), Host)
            host_red =
host.lookup_url('http://'+ip_local+':1275', Host)
            reducer = host_red.spawn('reducer',
'reducer/Reducer')

      server.readFile(hosts_maps,reducer,ip_server,fichero,slaves
,countingWords)
            serve_forever()
```

mapper.py

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



class Mapper(object):
      _tell = ['map']
      _ask = []
      _ref = ['map']


      def map(self,ip_server, num_archivo, reducer):
            dicc = {}
            palabras =
urllib.urlopen(ip_server+"/partes/file_"+str(num_archivo))
            texto = palabras.read()
            texto = texto.lower()
            texto= texto.replace('.', '').replace(',',
'').replace(':', '').replace(';', '').replace('\n', '
').replace('\r', ' ').replace('#', '').replace('[',
'').replace(']', '').replace('*','').replace('  ', '
').replace('-', ' ').replace('_', '').replace('?',
'').replace('!', '').replace('\'', ' ').replace('\"',
```

```
'').replace('(', '').replace(')', '').replace('=',
'').replace('<', '').replace('>', '')
          splits = texto.split(" ")
          for x in splits:
               if (x==''):
                    continue
               if x.endswith('-') or x.startswith('-'):
                    x.replace('-','')
               dicc[x] = dicc.get(x, 0) + 1
          reducer.trabaja(dicc)



if __name__ == "__main__":
     set_context()
     direccion = sys.argv[2]
     i = sys.argv[1]
     host = create_host('http://'+direccion+':130'+i)#creamos
un server para el mapper
     print "Mapper creado en http://"+direccion+":130"+i
     serve_forever()
```

reducer.py

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


global dicc
dicc={}


class Reducer(object):
     _tell = ['iniciar_tiempo', 'parar_tiempo', 'trabaja']
```

```
	_ask = []

	def iniciar_tiempo(self, slaves, programa):
		self.tiempoInicial = time.time()
		self.slaves = slaves
		self.programa = programa

	def parar_tiempo(self):
		tiempoFinal = time.time()
		tiempo = tiempoFinal - self.tiempoInicial
		print "Tiempo: ",tiempo



	def trabaja(self, palabras):
		global dicc
		for key in palabras.keys():
			dicc[key] = dicc.get(key, 0) + palabras[key]
		self.slaves=self.slaves-1
		if(self.slaves==0):
			if (self.programa==True):
				result = 0
				for key in dicc.keys():
					result = result +int(dicc[key])
				print "Counting Words: ",result
			else:
				result = 0
				print "Word count: \n"
				for key in dicc.keys():
					print str(key),":",dicc[key],"\n"
			self.parar_tiempo()

if __name__ == "__main__":
	set_context()
```

```
direccion = sys.argv[1]
host = create_host('http://'+direccion+':1275')
print "Reducer creado en http://"+direccion+":1275\n"
serve_forever()
```

secuencial.py

```
from pyactor.context import set_context, create_host, sleep,
shutdown
import sys, io, os, time


class Reducer(object):
        _tell = ['iniciar_tiempo', 'parar_tiempo', 'trabaja']
        _ask = []

        def iniciar_tiempo(self, slaves, programa):
                self.tiempoInicial = time.time()
                self.slaves = slaves
           self.programa = programa




def parar_tiempo(self):
                tiempoFinal = time.time()
                tiempo = tiempoFinal - self.tiempoInicial
                print "Tiempo: ",tiempo

def trabaja(self, palabras):
     dicc = {}
                for key in palabras.keys():
                        dicc[key] = dicc.get(key, 0) +
palabras[key]
                self.slaves=self.slaves-1
```

```
                if(self.slaves==0):
                        if (self.programa==True):
                                result = 0
                                for key in dicc.keys():
                                        result = result
+int(dicc[key])
                                print "Counting Words: ",result
                        else:
                                result = 0
                                print "Word count: \n"
                                for key in dicc.keys():
                                        print
str(key),":",dicc[key],"\n"
                        self.parar_tiempo()


class Mapper(object):
        _tell = ['map']
        _ask = []
        _ref = ['map']


        def map(self, archivo, reducer):
                diccionario = {}
           d = os.getcwd()
           path = d+"/texts/"+archivo
                palabras = io.open(path, "r", encoding="utf-8-
sig")
                texto = palabras.read()
                texto = texto.lower()
                texto= texto.replace('.', '').replace(',',
'').replace(':', '').replace(';', '').replace('\n', '
').replace('\r', ' ').replace('#', '').replace('[',
'').replace(']', '').replace('*','').replace('  ', '
').replace('-', ' ').replace('_', '').replace('?',
'').replace('!', '').replace('\'', ' ').replace('\"',
'').replace('(', '').replace(')', '').replace('=',
'').replace('<', '').replace('>', '')
```

```
                splits = texto.split(" ")
                for x in splits:
                        if (x==''):
                     continue
                if x.endswith('-') or x.startswith('-'):
                                x.replace('-','')
                        diccionario[x] = diccionario.get(x, 0) +
1
                reducer.trabaja(diccionario)



if __name__ == "__main__":
      set_context()
      direccion=str(sys.argv[1])
      fichero=str(sys.argv[2])
      if (str(sys.argv[3]) == "cw"):
                countingWords = True
        else:
                countingWords = False
      host = create_host('http://'+direccion+':2000')
        reducer = host.spawn('reducer', Reducer)
      host2 = create_host('http://'+direccion+':3000')
        mapper = host2.spawn('mapper', Mapper)
      reducer.iniciar_tiempo(1,countingWords)
      mapper.map(fichero,reducer)
      sleep(5)
      shutdown()
```

script_ini.py

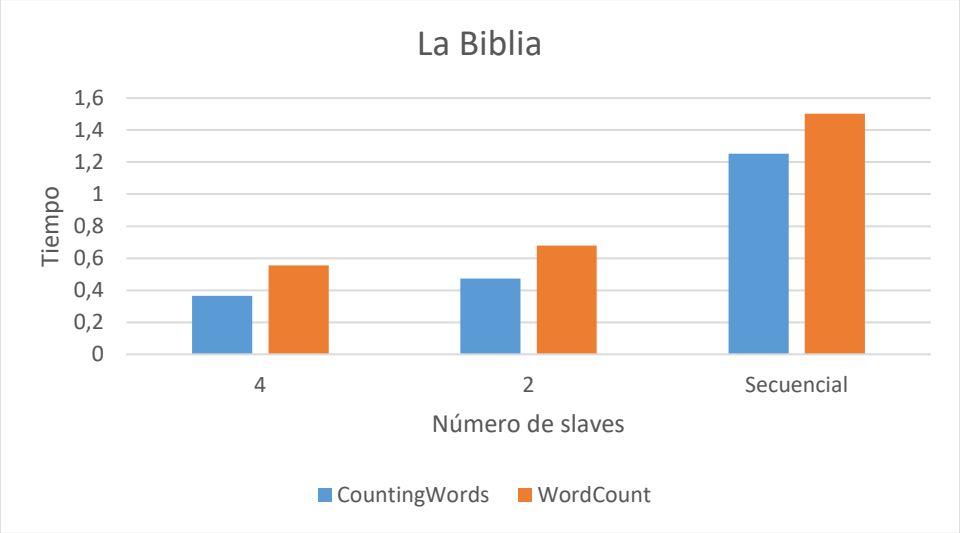
```
import os, sys


#    echo -e "1. Nombre fichero"

#    echo -e "2. Número slaves"

#    echo -e "3. Programa (CountingWords = cw || WordCount =
wc)"

#    echo -e "4. IP local"

#    echo -e "5. IP server"


#prints de prueba:

print "1. Nombre fichero: "+sys.argv[1]

print "2. Número slaves: "+sys.argv[2]

print "3. Programa (CountingWords = cw || WordCount = wc):
"+sys.argv[3]

print "4. IP local: "+sys.argv[4]

print "5. IP server: "+sys.argv[5]


slaves=int(sys.argv[2])

os.system("gnome-terminal -e 'bash -c \"python reducer.py %s %i;
exec bash\"'" %(sys.argv[4], slaves))

for i in range(slaves):

     os.system("gnome-terminal -e 'bash -c \"python mapper.py %i
%s; exec bash\"'" %(i,sys.argv[4]))

os.system("python server.py %s %i %s %s %s" %(sys.argv[1],
slaves, sys.argv[3], sys.argv[4], sys.argv[5]))
```

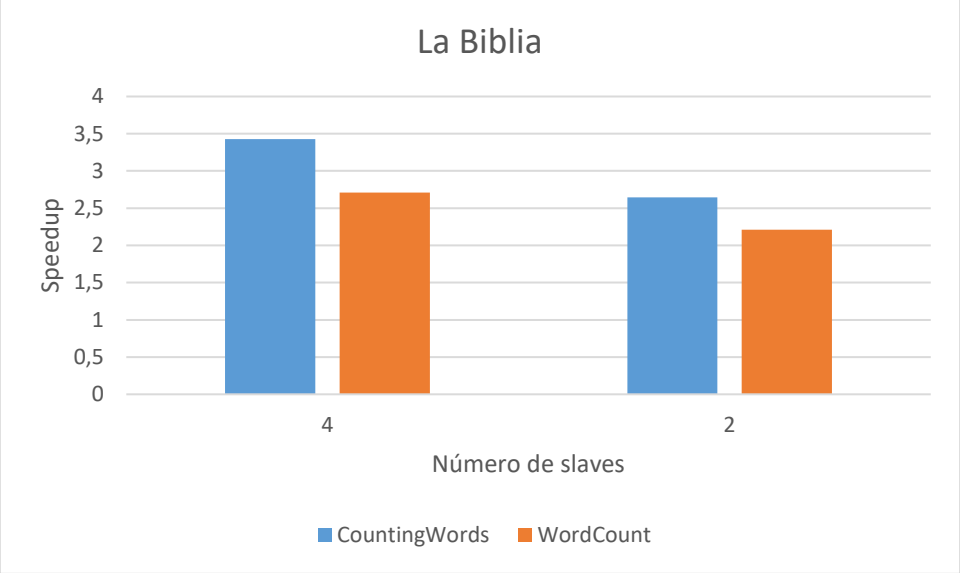
Juego de pruebas (speedup)

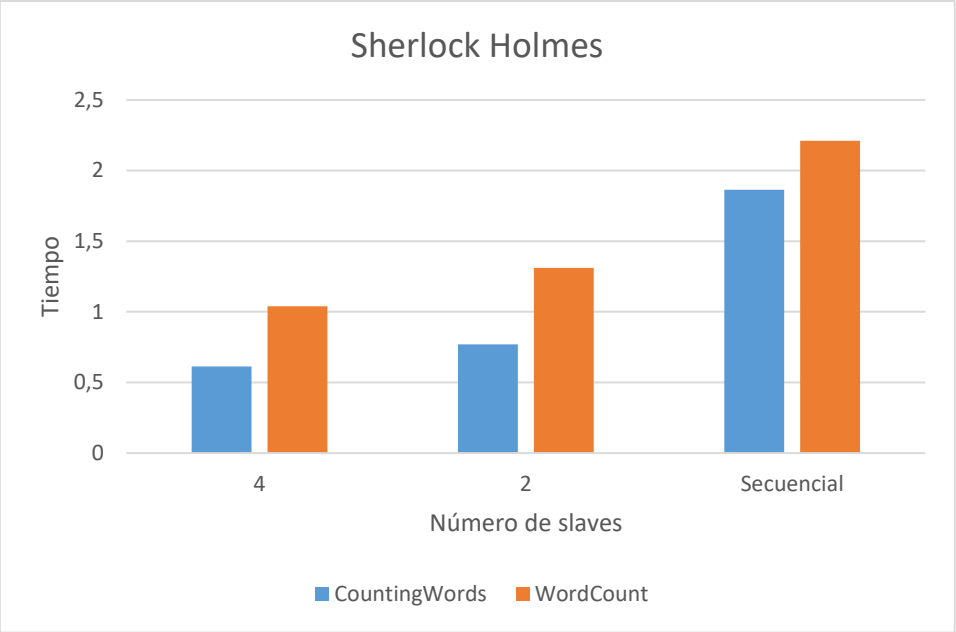
Nuestros portátiles tienen 4 cores, así que usaremos 4 slaves para aprovechar al máximo su capacidad. Además, también comprobaremos el tiempo usando sólo 2 cores para observar mejor el incremento progresivo. En el caso del secuencial, usaremos un solo slave y ejecutaremos un fichero que contiene tanto la función de master como la de los slaves (mapper/reducer).

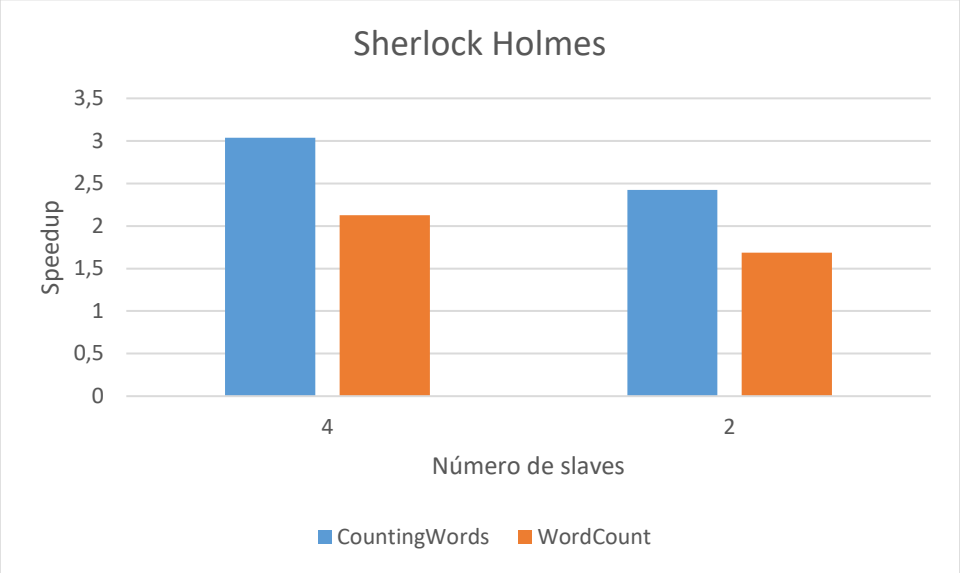
En el caso de "El Quijote" no hemos podido comprobarlo ya que nos daba un error debido a los acentos. Aun leyendo el fichero como utf-8 y codificando el proyecto también como tal, no hemos podido solucionarlo. El error es el siguiente:

```
<Fault 1: "<class 'xml.parsers.expat.ExpatError'>:not well-formed (invalid token): line 24, column 5">
```









Como se puede observar en las gráficas, a cuantos más cores se usan menos tiempo tarda el programa en ejecutarse.

También observamos que el CountingWords da mejores resultados que el WordCount. Esto puede ser debido al tiempo que tarda en imprimir por pantalla todas las claves, mientras que el CountingWords sólo tiene que sumar los valores de cada una de ellas.

Vemos que la reducción de tiempo de secuencial a 2 slaves es mayor que la de 2 a 4. Esta reducción se debe claramente a realizar el trabajo en distribuido y a que en distribuido se abren los archivos desde el servidor de files y en el secuencial directamente desde el directorio.

Nos hemos percatado que el abrir una file desde el directorio supone mayor tiempo que desde el servidor, con lo que el tiempo de secuencial presenta también la penalización de abrir los archivos desde el directorio.

En el caso del speedup, también podemos observar cómo entre CountingWords y WordCount, el primero es el que mejor speedup obtiene.

Conclusión

Una vez analizado el resultado del juego de pruebas, podemos concluir que usando dos cores en vez de uno solo ya nos da un buen speedup, pero que alcanza su máximo potencial (en nuestro portátil) al usar los 4 cores disponibles. Por lo tanto, es mucho más efectivo trabajar en distribuido que en secuencial.