Práctica final

A.C.S.I.

Grado en Ingeniería Informática, UIB. 2017.

Boutaour Sanchez, A. Antonio 43202227Q
Pericàs Serra, Bernat 43212796M

0.Índice

1. Introducción

El problema planteado consiste en monitorizar el sistema durante la realización de ciertas transferencias de un archivo para diferentes niveles de concurrencia. Esta tarea podría llevarse a cabo de manera manual pero el nivel de dificultad sería muy elevado, sobretodo debido a los diferentes niveles de concurrencia. Por este motivo conviene automatizarlo y, ya que el bash de linux ofrece muchas posibilidades, un script es lo más adecuado para realizar la solución del problema.

Son cinco las tareas a monitorizar, nótese que todas realizan una transferencia del punto A al punto B siendo el segundo el disco duro del ordenador elegido: transferir un archivo desde otra carpeta del disco duro del ordenador, desde un pendrive, desde otro dispositivo (en la misma red wifi), desde un servidor próximo y, finalmente, desde un servidor geográficamente lejano.

El ordenador escogido es un portátil Acer Aspire V5 con las siguientes características:

- 8 GB de RAM
- CPU Intel® Core™ i7-4500U CPU @ 1.80GHz de 4 núcleos
- Disco duro SSD 500GB Samsung
- Ubuntu 17.04
- Temperatura media de la CPU durante las pruebas: 48ºC

Las características de la red wifi son:

- Fibra simétrica de Movistar
- Velocidad de descarga: 31.9 Mbps
- Velocidad de subida: 29.7 Mbps

Finalmente, las características del entorno son:

- Fecha: 10 de junio del 2017
- Temperatura: 26ºC
- Localización: Palma, Mallorca

2. Tabla de comandos

A continuación se muestra una tabla con las métricas evaluadas y el comando usado para cada una de ellas. Nótese que hay más métricas que comandos, ya que los comandos elegidos proporcionan más de una métrica.

Métrica	Comando
CPU	**top**
Cantidad de RAM	**top**
Velocidad lectura HD	**iotop**
Velocidad escritura HD	**iotop**
Velocidad de descarga	**wget**
Tiempo de ejecución total	**time**
Tiempo de CPU usuario	**time**
Tiempo de CPU sistema	**time**

top
Este comando es seguramente el más conocido en cuanto a monitorización del sistema ya que proporciona una lista de los procesos que se están ejecutando, con diversas métricas como el estado de la CPU usada por este, la memoria, su pid, etc.

iotop
Este otro comando no viene instalado por defecto en ubuntu pero se puede instalar en cuestión de segundos (sudo apt install iotop), una vez instalado y ejecutado con privilegios, muestra, entre otras métricas, las principales velocidades de lectura y escritura del disco duro, así como las de cada proceso que está ejecutándose.

Durante la realización de pruebas de cada script, nos dimos cuenta que el comando *iotop* no parecía mostrar resultados fiables siempre, ya que algunas veces las velocidades escritura eran igual a 0, hecho inasumible ya que, si la copia del archivo se realizaba, es imposible que la velocidad sea 0[1].

Finalmente descubrimos el motivo, y es que este comando almacena datos en caché y los resultados de las velocidades son 0. Para evitarlo, cada vez que se realiza una transferencia del archivo, el destino de este debe estar en un directorio diferente. Motivo por el cual creamos un directorio llamado *d* en el cual hay 4 directorios: *d1*, *d2*, *d3* ,*d4*, uno para cada nivel de concurrencia.

[1] Esto retrasó la finalización del script ya que conllevó una tarea de investigación y experimentos extra para determinar qué podía estar pasando.

En cambio, las velocidades de lectura no pudieron ser obtenidas de los 3 últimos experimentos, sólo cuando se ejecuta el script para monitorizar la transferencia des del mismo computador o desde un pendrive, y en este caso no pudimos averiguar el motivo.

wget

Se usa el *wget* para la velocidad de descarga ya que, al final de cada transferencia, muestra la media de velocidad de descarga, evitando así tener que calcular la media manualmente.
En caso de tratarse de una monitorización de un servidor, es necesario haber instalado la herramienta de servidores de apache (sudo apt install apache2) y que se esté ejecutando (apache server start).

time

El comando *time* sirve para obtener los tiempos de usuario, sistema y total (o real) de una ejecución, sea un comando, script o programa. Este se usará en la instrucción clave del script, por ejemplo, en el copy en caso de las dos primeras ejecuciones. Simplemente, el comando a monitorizar será encapsulado en el comando time, a la vez que la salida de este se almacena en un log.

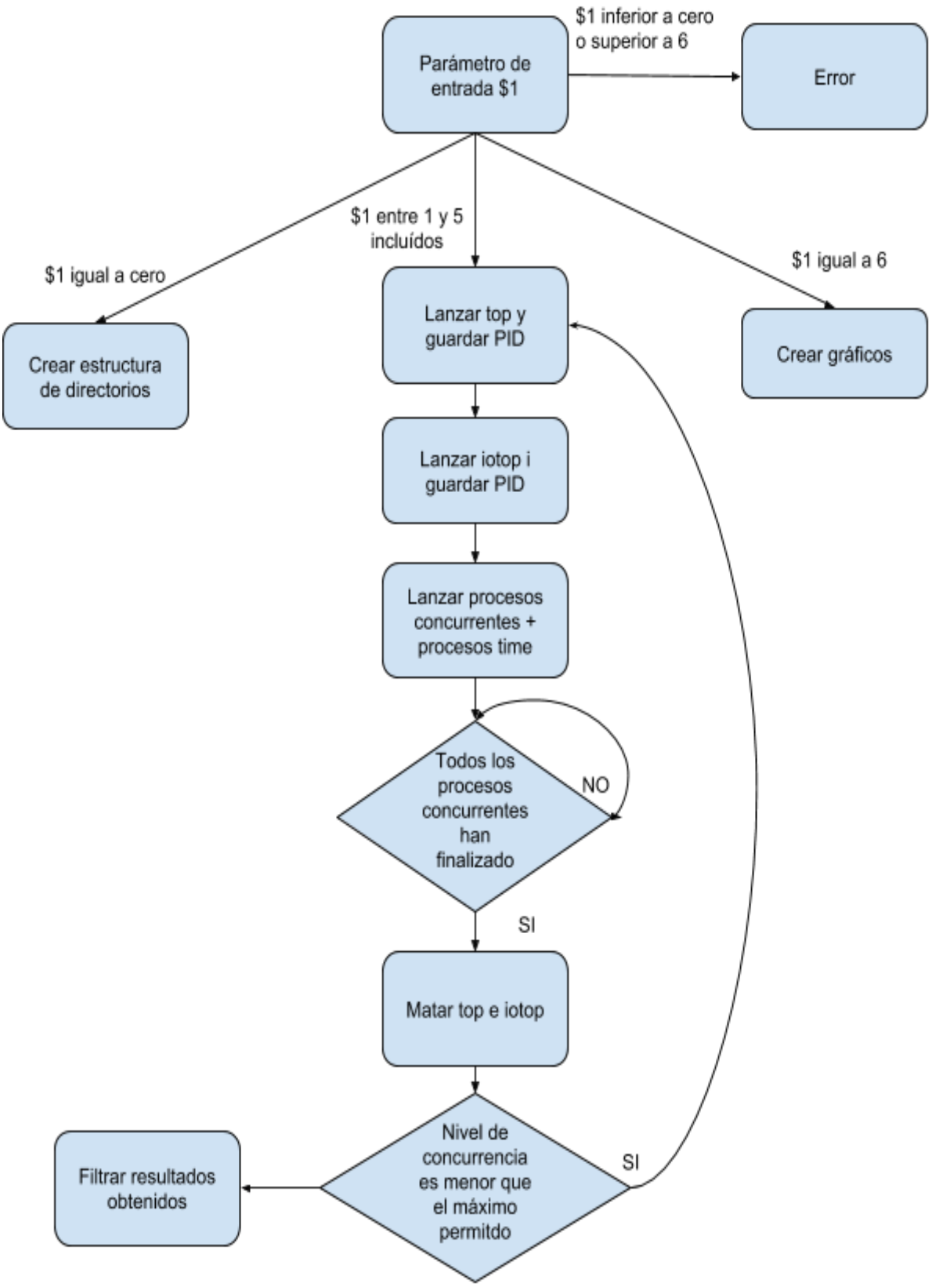
3. Scripts y logigramas

En un principio, realizamos varios scripts, uno para cada tarea a monitorizar, pero nos dimos cuenta de que prácticamente eran idénticos. La única diferencia entre ellos era, obviamente, el comando usado para realizar la transferencia, que cambiaba en cada caso. Así que decidimos realizar todas las tareas en un único script y que el comando clave dependiera de una variable parametrizada.

A parte de este script, creamos dos más, uno para crear la estructura de directorios necesaria para almacenar los logs y otro que realizaba los gráficos con *gnuplot.* Una vez acabados todos los scripts necesarios para la realización de la práctica, decidimos incluirlos también en el script principal, quedando un único script que, dependiendo de un valor introducido por línea de comandos, puede crear la estructura de directorios (primer paso), realizar y monitorizar 5 tareas individualmente (segundo paso) y, finalmente, crear los gráficos para interpretar los resultados (tercer paso). Optamos por esta solución para dar más control al usuario y saber en cada momento qué tarea está realizando. Además, aunque toda función necesaria para la prçactica pueda ser realizada por el script principal, los otros scripts siguen presentes, es decir, no se ha incluido el código en sí, sino que des del script padre se llama a los scripts para crear la estructura de directorios y los gráficos.

Es importante saber que para realizar un paso los pasos anteriores deben haber sido realizados al menos una vez. Por ejemplo, no tiene sentido monitorizar alguna tarea sin haber creado previamente el árbol de directorios, o realizar los gráficos sin tener los resultados.

A continuación se muestra el logigrama del script final que permite ejecutar todas las funciones de la práctica.



A continuación se muestra el código del script principal (dividido en dos páginas por cuestiones de espacio), así como el de los scripts de los cuales depende.

```
#!/bin/bash
#Práctica final ACSI. Bernat Pericàs Serra i Antonio Boutaour Sanchez. 2017.

M=$1            #número de script/servicio monitorizado (depende del parámetro de entrada)
MB=$M   #guardamos el valor de M ya que cambiará durante la ejecución
echo "-------$M.sh-------"
MIN=1       #minima concurrencia
MAX=4       #máxima concurrencia
N=$MIN      #concurrencia actual (valor inicial=minimo nivel de concurrencia deseado)
PIDS=()     #array de pids de procesos concurrentes que se monitorizaran
PID_TOP=0   #pid del top que monitoriza los procesos concurrentes
PID_IOTOP=0 #pid del iotop que monitoriza los procesos concurrentes
#variables para contar números de líneas (para hacer medias)
L=0         #número de veces que aparece el comando (TOP)
LL=0        #número de veces que aparece el comando (IOTOP)
LLL=0       #número de veces que aparece el comando (WGET)
LLLL=0      #número de veces que aparece el comando (TIME)
ARCHIVO=ubuntu.iso      #nombre del archivo con el que se hacen las pruebas
ADDRESS=192.168.1.54/ubuntu.iso     #dirección para la prueba M=3 i M=4
URL=https://www.dropbox.com/s/zn3q6rk85lz4v7v/ubuntu.iso?dl=1 #link al archivo para (prueba M=5)

if ((MB>=0))&&((MB<=6)); then
    if ((MB==0)); then
        #crear estructura de directorios necesaria
        ./0.sh
    fi
    if ((MB>=1))&&((MB<=5)); then   #si M vale 1,2,3,4 o 5
        while [[ (($N -le $MAX)) ]]; do #recorrer niveles de concurrencia
            #lanzar procesos que monitorizan
            echo "Concurrencia: $N"
            top -b -d 1 > outputs/m$M/c$N/top_output.log &
            PID_TOP=$!

            iotop -b -P -q -k > outputs/m$M/c$N/iotop_output.log &
            PID_IOTOP=$!

            echo "logs de tiempo" > outputs/m$M/c$N/time_output.log

            #lanzar los procesos concurrentes que seran monitorizados
            #tambien se ejecuta el comando "time" en cada ejecución
            for (( i = 1; i < ($N+1); i++ )); do
                #ejecuta un comando u otro dependiendo de M
                case $M in
                    1) { /usr/bin/time -f "\t%e r,\t%U u,\t%S s"
                        cp $ARCHIVO d/d$i/; }
                        2>> outputs/m$M/c$N/time_output.log &;;

                    2) { /usr/bin/time -f "\t%e r,\t%U u,\t%S s"
                        cp /media/bernat/some_label/$ARCHIVO d/d$i/; }
                        2>> outputs/m$M/c$N/time_output.log &;;

                    3) { /usr/bin/time -f "\t%e r,\t%U u,\t%S s"
                        wget -o outputs/m$M/c$i/wget.log $ADDRESS -P d/d$i; }
                        2>> outputs/m$M/c$N/time_output.log &;;

                    4) { /usr/bin/time -f "\t%e r,\t%U u,\t%S s"
                        wget -o outputs/m$M/c$i/wget.log $ADDRESS -P d/d$i; }
                        2>> outputs/m$M/c$N/time_output.log &;;

                    5) { /usr/bin/time -f "\t%e r,\t%U u,\t%S s"
                        wget -o outputs/m$M/c$i/wget.log $URL -P d/d$i; }
                        2>> outputs/m$M/c$N/time_output.log &;;
                esac
                PIDS[${#PIDS[@]}]=$!    #guardar pid del comando lanzado
            done
            sleep 1 #esperar 1 segundo para prevenir errores

            #esperar a que acaben todos los procesos concurrentes
            for i in "${PIDS[@]}"; do wait $i; done
```

```
            #matar todos los procesos que monitorizan
            disown $PID_TOP; kill -9 $PID_TOP
            disown $PID_IOTOP; kill -9 $PID_IOTOP
            #borrar array
            unset PIDS
            ((N++)) #incrementar N para pasar al siguiente nivel de concurrencia
        done
        #tratar resultados
        echo "CPU   RAM READ    WRITE   DOWN    TOTAL_T USER_T  SYSTEM_T"
            > outputs/m$M/resultados_$M.txt #header del fichero
        N=$MIN  #reiniciamos la variable N al valor inicial para volver a hacer una iteración
        while [[ (($N -le $MAX)) ]]; do
            #contar numero de veces que aparece el proceso que se monitoriza (cp en este caso)
            L="$(cat outputs/m$M/c$N/top_output.log | grep -w "cp" | wc -l)"    #lineas top
            LS=$(($L%$N))   #lineas sobrantes
            CPU="$(cat outputs/m$M/c$N/top_output.log | grep -w "cp" | head -n -$LS |
                awk -v N=$N '{C+=$9} END {print C/(NR/N)}')"
            RAM="$(cat outputs/m$M/c$N/top_output.log | grep -w "cp" | head -n -$LS |
                awk -v N=$N '{C+=$6} END {print C/(NR/N)}')"

            LL="$(cat outputs/m$M/c$N/iotop_output.log | grep -w "cp" | wc -l)"
            READ="$(cat outputs/m$M/c$N/iotop_output.log | grep -w "cp" |
                awk -v LL=$LL '{X+=$4} END {print X/LL}')"
            WRITE="$(cat outputs/m$M/c$N/iotop_output.log | grep -w "cp" |
                awk -v LL=$LL '{X+=$6} END {print X/LL}')"

            if ((M=1))||((M=2)); then DOWN=-1 #1 y 2 no requieren velocidad de descarga
            else
                LLL="$(cat outputs/m$M/c$N/wget.log | grep "saved" | wc -l)"
                DOWN="$(cat outputs/m$M/c$N/wget.log | grep "saved" | sed 's/(//g' |
                    awk '{D+=$3} END {print D/$LLL}')"
            fi

            LLLL="$(cat outputs/m$M/c$N/time_output.log | wc -l)"
            TOTAL_T="$(cat outputs/m$M/c$N/time_output.log | awk '{T+=$1} END {print T/$LLLL}')"
            USER_T="$(cat outputs/m$M/c$N/time_output.log | awk '{T+=$3} END {print T/$LLLL}')"
            SYSTEM_T="$(cat outputs/m$M/c$N/time_output.log | awk '{T+=$5}END {print T/$LLLL}')"

            #escribir los valores de las variables en el fichero de los resultados
            echo "$CPU  $RAM    $READ   $WRITE  $DOWN   $TOTAL_T    $USER_T $SYSTEM_T"
                >> outputs/m$M/resultados_$M.txt
            ((N++)) #se incrementa N para realizar la iteración de la siguiente concurrencia
        done
    fi
    if ((MB==6)); then
        #crear plots a partir de los resutados obtenidos
        ./gnuplot.sh
    fi
else
    echo "Parámetro de entrada no válido. $1 entre [0,6]"
fi
```

A continuación, el código del script “0.sh”, encargado de crear el árbol de directorios:

```
#!/bin/bash
#borra y crea directorios necesarios

if [[ -d outputs ]]; then rm -rf outputs; fi
mkdir outputs outputs/PLOTS/
if [[ -d d ]]; then rm -rf d; fi
mkdir d
for (( i = 1; i <= 5; i++ )); do
    mkdir outputs/m$i d/d$i
    for (( j = 1; j <= 4; j++ )); do
        mkdir outputs/m$i/c$j
    done
done
rm -rf d/d5
echo "Árbol de directorios creado"
```

Finalmente, el script “gnuplot”:

```
#!/bin/bash
#5.1 y 5.3
for (( i = 1; i <= 5; i++ )); do #recorre archivos
    cat outputs/m$i/resultados_$i.txt | tail -n 4 > outputs/m$i/res_$i.txt
    for (( j = 1; j <= 4; j++ )); do #recorre lineas
        read -r LINE
        X="$(echo $LINE | awk '{print $4}')"
        echo "WRITE_$j: $X" >> outputs/m$i/WRITE_m$i.txt
    done < outputs/m$i/res_$i.txt
    gnuplot -e "IN_FILE='outputs/m$i/WRITE_m$i.txt';
    OUT_FILE='outputs/PLOTS/WRITE_m$i.png'" gnuplot_WRITE.gnuplot

    for (( j = 1; j <= 4; j++ )); do #recorre lineas
        read -r LINE
        X="$(echo $LINE | awk '{print $1}')"
        echo "CPU_$j:   $X" >> outputs/m$i/CPU_m$i.txt
    done < outputs/m$i/res_$i.txt
    gnuplot -e "IN_FILE='outputs/m$i/CPU_m$i.txt';
    OUT_FILE='outputs/PLOTS/CPU_m$i.png'" gnuplot_CPU.gnuplot
done

#5.2
#arrays de concurrencias: C= concurrencia 1, CC= concurrencia 2, etc
for (( i = 1; i <=5; i++ )); do #recorre ficheros
    for (( j = 1; j <=1; j++ )); do #lee un fichero
        read -r LINE
        C[$i]="$(echo $LINE | awk '{print $5}')"
        read -r LINE
        CC[$i]="$(echo $LINE | awk '{print $5}')"
        read -r LINE
        CCC[$i]="$(echo $LINE | awk '{print $5}')"
        read -r LINE
        CCCC[$i]="$(echo $LINE | awk '{print $5}')"
    done < outputs/m$i/res_$i.txt
done
#transformar arrays en ficheros
for (( j = 1; j <=5; j++ )); do echo "DOWN_$j:  ${C[$j]}">>outputs/DOWN_c1.txt; done
for (( j = 1; j <=5; j++ )); do echo "DOWN_$j:  ${CC[$j]}">>outputs/DOWN_c2.txt; done
for (( j = 1; j <=5; j++ )); do echo "DOWN_$j:  ${CCC[$j]}">>outputs/DOWN_c3.txt; done
for (( j = 1; j <=5; j++ )); do echo "DOWN_$j:  ${CCCC[$j]}">>outputs/DOWN_c4.txt; done
#crear gnuplots
for (( i = 1; i <= 4; i++ )); do
    gnuplot -e "IN_FILE='outputs/DOWN_c$i.txt';
    OUT_FILE='outputs/PLOTS/DOWN_c$i'" gnuplot_DOWN.gnuplot
done

echo "Gnuplots creados"
```

El cual depende de 3 ficheros que contienen el código necesario para la creación de los tres tipos de gráficas:

Para la velocidad de escritura:

```
set terminal pngcairo font "arial,10" size 500,500
set output OUT_FILE
set boxwidth 0.75
set style fill solid
set xlabel 'Nivel de concurrencia'
set ylabel 'Velocidad de escritura(Kb/s)'
set key off
set yrange [0:50000]
plot IN_FILE using 2:xtic(1) with boxes
```

Para la velocidad de descarga:

```
set terminal pngcairo font "arial,10" size 500,500
set output OUT_FILE
set boxwidth 0.75
set style fill solid
set xlabel 'Localización'
set ylabel 'Velocidad de descarga(Mbps)'
set key off
set yrange [0:60]
plot IN_FILE using 2:xtic(1) with boxes
```

Y, finalmente, para la distribución de CPU:

```
set terminal pngcairo font "arial,10" size 500,500
set output OUT_FILE
set boxwidth 0.75
set style fill solid
set xlabel 'Nivel de concurrencia'
set ylabel 'CPU usada(%)'
set key off
set yrange [0:100]
plot IN_FILE using 2:xtic(1) with boxes
```

4. Resultados

En la siguiente tabla se muestran los resultados obtenidos después de la ejecución del script, hay celdas sin datos ya que no tiene sentido monitorizar algunas métricas con según qué transferencias. Concretamente, la velocidad de descarga de un archivo no se puede evaluar si este está siendo transferido a través de internet.
Por cuestiones de espacio, se ha dividido la tabla en dos partes.

Métrica	HD				USB			
Concurrencia	1	2	3	4	1	2	3	4
CPU	10.23	12.98	22.25	15.6	7.34	6.97	20.02	23.86
RAM	670.2	1547	2068	3458	1276.22	1760.32	2786.67	4037.78
READ_V	355547.63	26257.63	17513.75	16356	29228.14	26849.24	20495.25	13578.36
WRITE_V	39936.37	34653.29	25899.96	22003.65	48534.12	39547.56	30563.45	18378.57
DOWNLOAD_V	-	-	-	-	-	-	-	-
TOTAL_T	5.58	7.52	9.65	12.54	13.47	12	20.65	35.45
USER_T	0.04	0.06	0.05	0.1	0.14	0.16	0.07	0.07
SYSTEM_T	0.89	0.86	0.68	0.58	2.58	1.37	1.12	1.02

Métrica	Wifi				Servidor				Cloud			
Concurrencia	1	2	3	4	1	2	3	4	1	2	3	4
CPU	11.14	26.85	29.44	32.25	21.78	20.14	25.36	32.44	5.08	15.82	14.89	19.92
RAM	1859.22	2521.36	8569.71	7854.12	1785	3569.28	5771.82	6854.35	1960	3920.25	4250.93	4958.69
READ_V	0	0	0	0	0	0	0	0	0	0	0	0
WRITE_V	14524.47	10452.35	8654.45	7514.94	30084.46	276589.58	22150.47	23258.38	20067.45	17023.56	18998.46	16543.23
DOWNLOAD_V	25.7	22.7	23.78	17.85	48.1	35.15	36.58	41.69	2.65	1.74	2.55	2.84
TOTAL_T	67.44	64.5	65.48	129.78	35.54	45.69	37.57	42.43	618.58	850.71	630.74	615.15
USER_T	1.96	2.77	3.74	5.14	2.29	1.31	2.3	3.4	1.72	2.67	2.74	3.81
SYSTEM_T	22.1	25.55	30.58	31.74	10.29	9.19	10.02	11.66	31.96	30.62	30.73	32.72

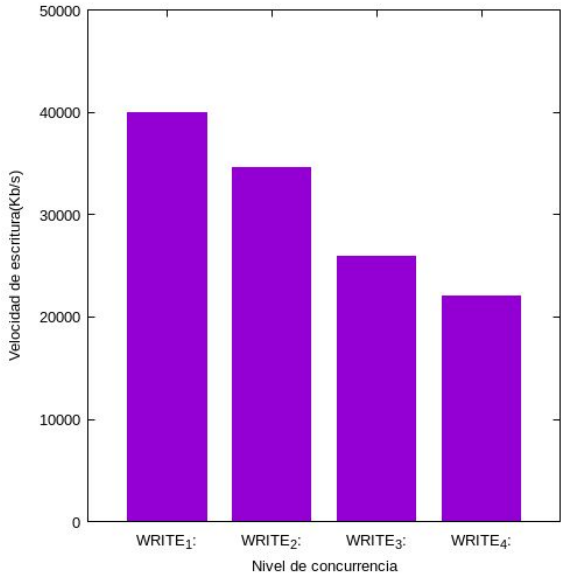
5. Gráficas

A continuación se muestran las gráficas generadas mediante *gnuplotting* y los datos obtenidos.

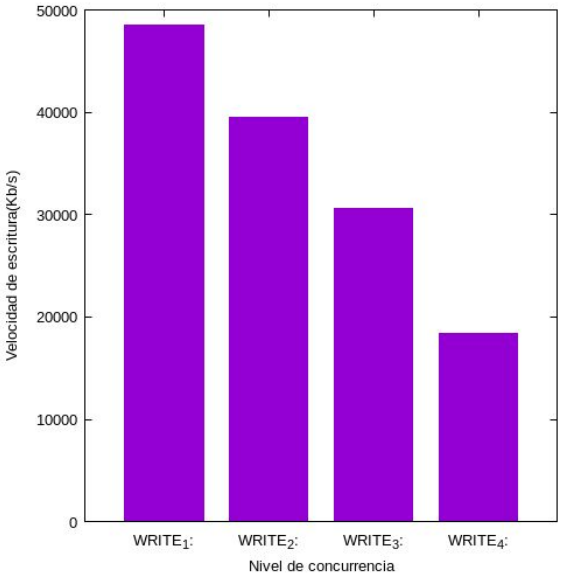
5.1. Velocidad de escritura - grado de concurrencia

Gráficas que muestran cómo cambia la velocidad de escritura del disco duro a medida que aumenta el grado de concurrencia. Una gráfica para cada uno de los cinco tipos de localizaciones de archivo.

Experimento 1

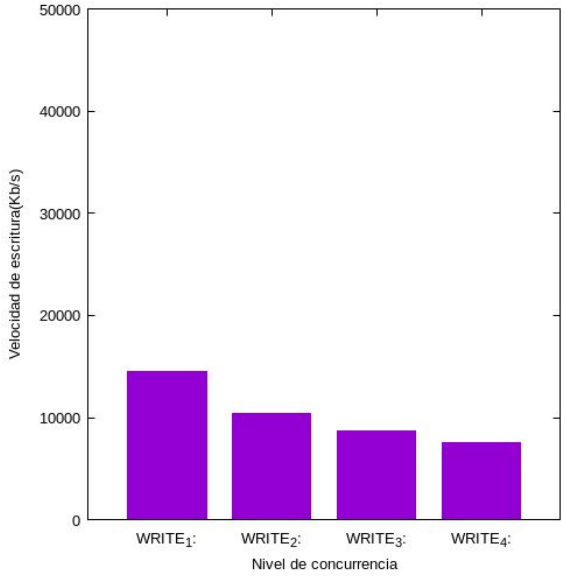


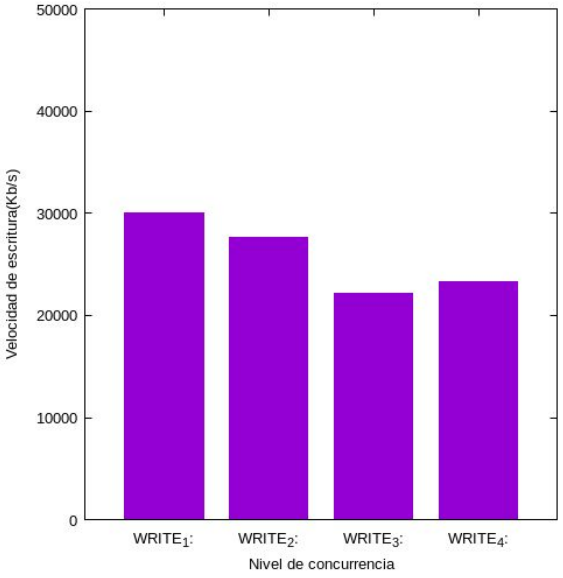
Experimento 2



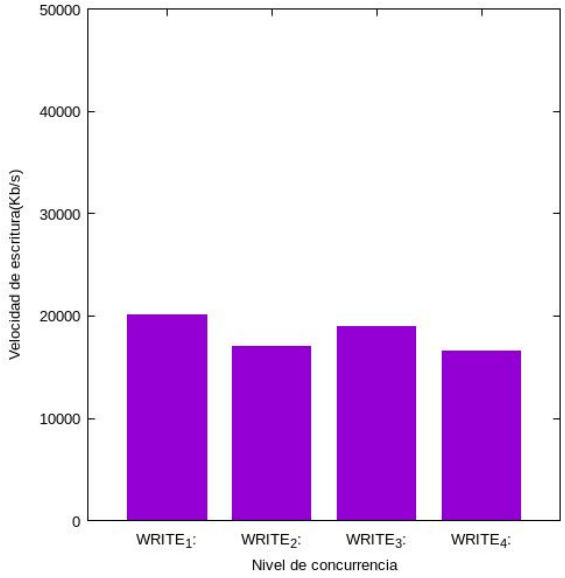
Experimento 3

Experimento 4





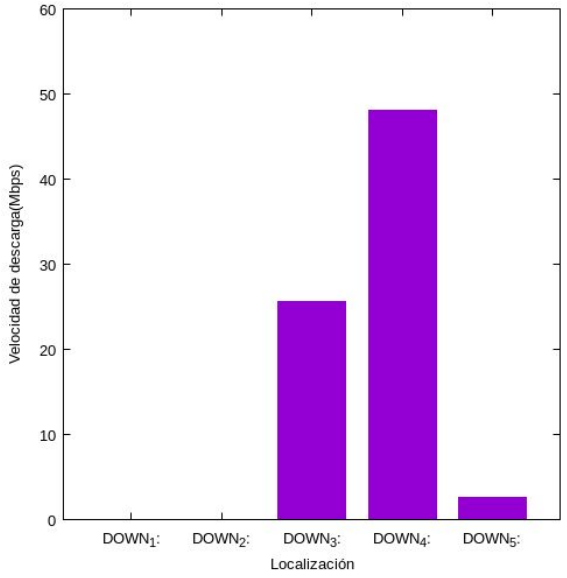
Experimento 5



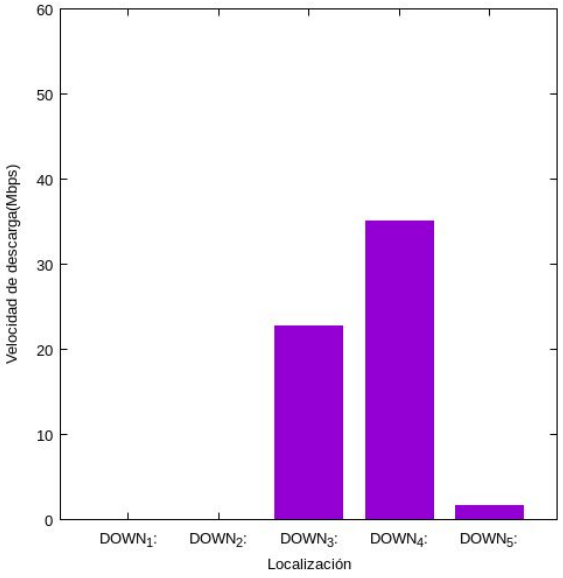
5.2. Uso de red - localización del archivo

Gráficas que muestran cómo cambia el uso de la red a medida que cambiamos la localización del archivo. Una gráfica distinta para cada nivel de concurrencia.

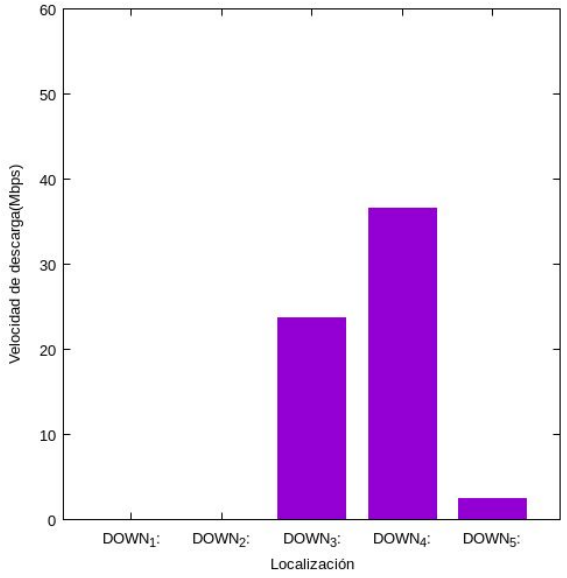
Localización 1



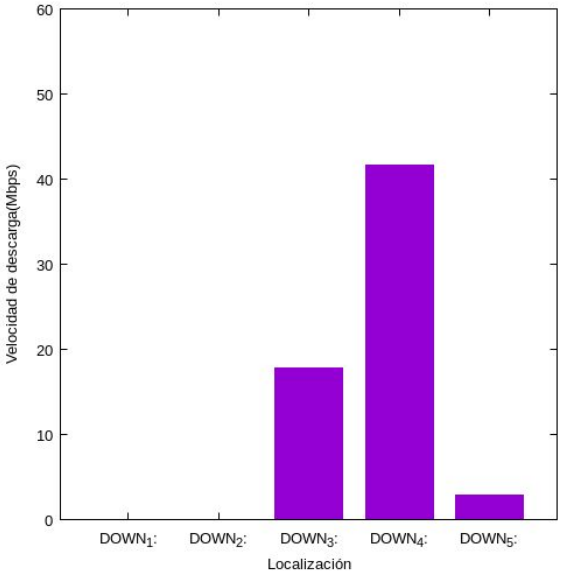
Localización 2



Localización 3



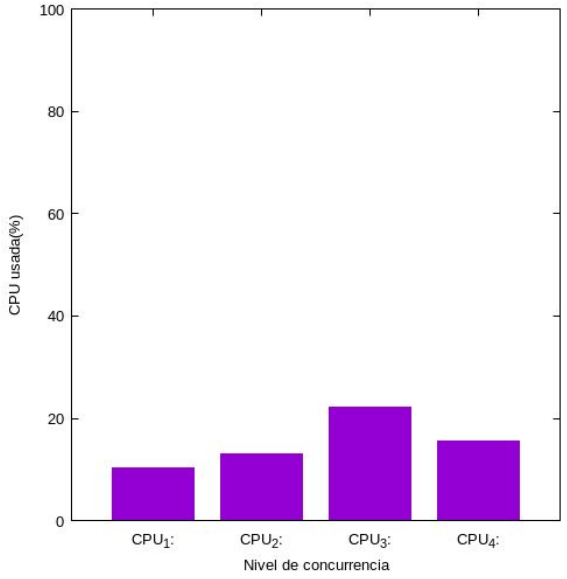
Localización 4



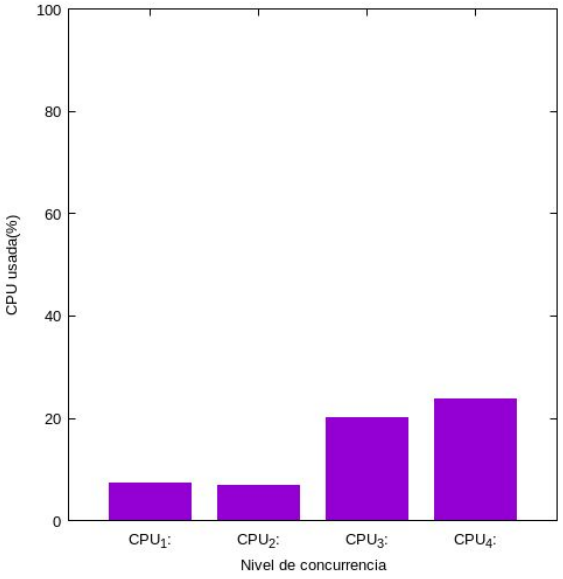
5.3. Distribución de CPU - grado de concurrencia

Gráficos que muestran cómo varía la distribución de CPU de usuario y sistema a medida que se incrementa el grado de concurrencia.

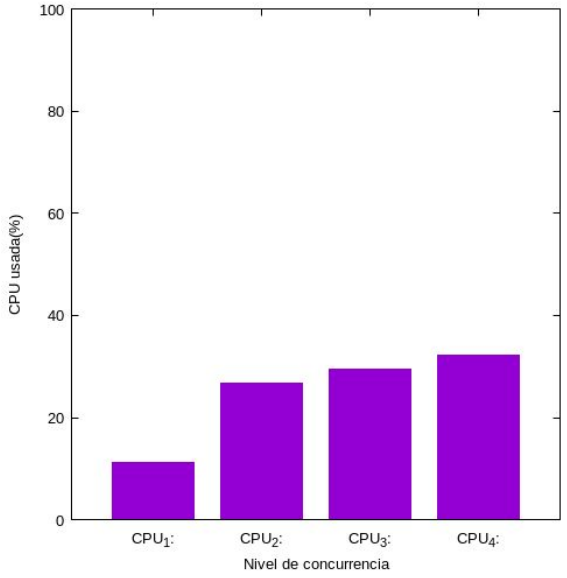
Experimento 1



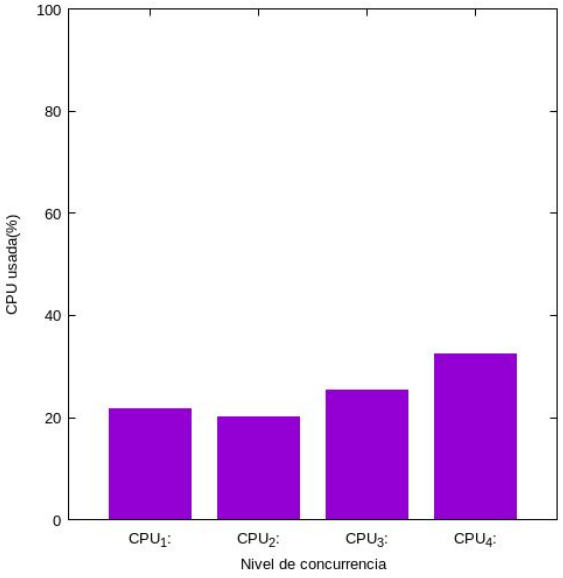
Experimento 2



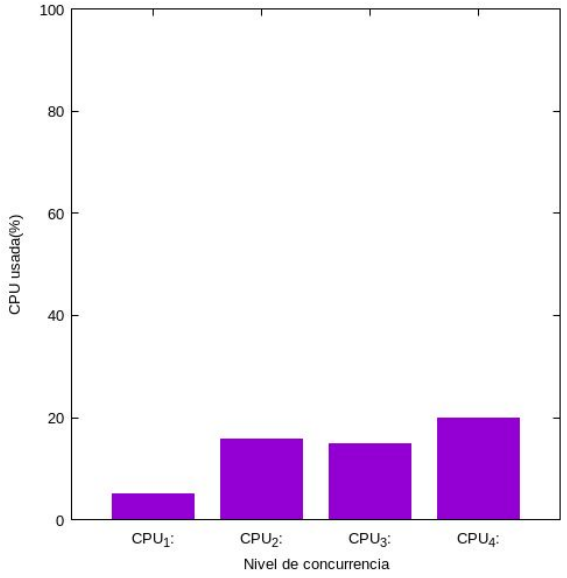
Experimento 3



Experimento 4



Experimento 5



6. Reflexiones

6.1 ¿Influye la localización del fichero en la memoria principal utilizada en el proceso de transferencia?

Observando los resultados obtenidos, podemos afirmar que sí influye, ya que, para los dos primeros experimentos, la media de RAM usada (entre todas las concurrencias) es de aproximadamente 1900 Mb y 2400 Mb, mientras que para los 3 experimentos siguientes, las medias son de 5200 Mb, 4495 Mb y 3772 Mb respectivamente. Es fácilmente observable como el uso de RAM aumenta. Esto puede deberse a que la transferencia de archivos por red es más complicada que la transferencia por otros medios, ya que los paquetes no llegan al computador de manera completamente ordenada y es necesario guardar información y archivos temporales durante la misma descarga, pero una vez acabada, estos son eliminados.

6.2 ¿Influye el nivel de concurrencia en el tiempo total de ejecución? ¿Y en el tiempo de cpu de usuario utilizado? ¿Y en el tiempo de CPU de sistema utilizado?

Como podemos ver en las tablas de los resultados obtenidos aparece una progresión en los tres tipos diferentes de tiempos. Podemos observar cómo a medida que avanza la carga del trabajo debido a los diferentes tipos de pruebas aumenta significativamente el tiempo total de ejecución, apreciándose una clara diferencia entre el tiempo de copia de la primera prueba (12.54s en nivel 4 de concurrencia) hasta llegar a la descarga del fichero en la nube, donde el tiempo de ejecución en nivel 4 de concurrencia llega 615,15s.

Una vez observado dicho aumento cabe destacar el incremento del tiempo total de ejecución en la misma prueba a distintos niveles de concurrencia. Este aumento se mantiene de una forma semi-lineal en la mayoría de pruebas, sin embargo cabe remarcar algunas anomalías surgidas, donde se puede ver que en algunas pruebas el tiempo de ejecución no ha aumentado con el nivel de concurrencia**. Deducimos que lo acontecido se debe a que a la hora de realizar las pruebas, en algunas de ellas el ordenador se encontraba con un uso de CPU mayor que al realizar la misma a diferente nivel de concurrencia, de manera que podía realizar el trabajo en un tiempo de ejecución menor.**

El tiempo total de usuario en este caso se puede observar de una forma más clara que el aumenta a medida que incrementa el nivel de concurrencia. Este hecho se produce en todas las pruebas, donde el tiempo total de usuario menor se establece en la prueba nº 1, mientras que en la ejecución del script para realizar la descarga de la nube aumenta el tiempo total de usuario hasta los 3.81 segundos a nivel de concurrencia 4.

Por último vemos el tiempo de CPU de sistema utilizado. En este último caso podemos observar por las pruebas realizadas que el nivel de concurrencia no afecta en este. Para

realizar la negación nos hemos basado en el hecho de que en en la mayoría de los resultados vemos que a nivel de concurrencia 4 el tiempo de CPU de sistema dedicado al proceso es menor que en el caso de nivel de concurrencia 1[2].

6.3 ¿Influye la localización del archivo en la velocidad de escritura en el disco duro?

Se puede observar como sí influye, en los experimentos donde el fichero está ubicado en un lugar materialmente conectado con el destino de este (experimentos uno y dos), las velocidades de escritura son muy elevadas, prácticamente alcanzando los 50000 Kb/s. En estos experimentos el cuello de botella es el dispositivo más lento de los dos (o el USB o el mismo disco del ordenador), y en ambos casos las velocidades de escritura y lectura son muy altas ya que ambos son de tecnología flash. En cambio, en caso de realizarse la transferencia por internet, las velocidades de transferencia son muy menores a las de escritura del archivo. Por este motivo, y basándonos en los resultados, podemos deducir como sí influye.

6.4 ¿Es la transferencia de ficheros un tipo de carga que demande unos altos niveles de CPU? ¿Se mantiene este comportamiento independientemente del nivel de concurrencia y del lugar de almacenamiento?

La transferencia de ficheros demanda cierta CPU, sobretodo para niveles de concurrencia altos, aunque no es nada alarmante ya que lo que el usuario suele querer es que la transferencia se realice en el menor tiempo posible. Por tanto, no se limita la CPU dedicada a esos procesos.
En cuanto al lugar de almacenamiento, se puede observar como en los experimentos de transferencia por red la demanda de CPU es ligeramente mayor, aproximadamente del 23%, frente al 15% de las transferencias locales.

6.5 A la vista de los resultados, ¿cuál crees que es el cuello de botella del sistema? ¿Es el mismo en todos los casos?

Basándonos en los resultados obtenidos, creemos que el cuello de botella es, para el primer experimento, obviamente el propio disco duro. Para el segundo experimento, el cuello de botella es el dispositivo USB ya que los tiempos totales aumentan considerablemente frente a los del experimento anterior.
Para los experimentos de transferencia inalámbrica, el cuello de botella es la propia red. Las velocidades de transferencia por red dependen fundamentalmente de las características del servidor donde se aloja el archivo y del ancho de banda de la red que se usa.

[2] Estos resultados se observan en en las pruebas realizadas en el HD, USB, Servidor y Cloud, mientras que en la prueba realizada con la red Wifi crece de manera lineal, hecho del cual se desconoce el motivo.

6.6 Teniendo en cuenta todos los resultados obtenidos, ¿dónde recomendarías tener almacenados los archivos con los que trabajamos habitualmente?

Es bien sabido que el hecho de teniendo almacenados todas nuestras fotos y videos en la nube podemos obtener dichos datos en diferentes dispositivos y de maner mucho más portable que en los diferentes sistemas que le precedieron. No obstante podemos remarcar de manera importante que la nube es eficiente a nivel de almacenamiento para aquellos archivos con los que trabajamos con los que no se realiza un trabajo habitual, como hemos remarcado anteriormente hablamos de datos del tipo multimedia[3]. Para archivos con los que tenemos que realizar un uso habitual lo más eficiente es tenerlo almacenado en el disco duro, siendo este el que ofrece mejores servicios en lectura/escritura y velocidad de transferencia.
Destacamos que hemos realizado esta recomendación si se tiene en cuenta los resultados obtenidos en las pruebas y que el usuario puede trabajar habitualmente en el mismo sistema. En caso de que otro tipos de necesidades nuestra recomendación se puede ver afectada. Un claro ejemplo es que si el usuario utiliza varios terminales con asiduidad recomendamos que utilice el sistema de almacenamiento USB, puesto que ofrece la mejor relación de velocidad/comodidad que los otros tipos de almacenamiento.

6.7 ¿Puedes asegurar que tus datos son fiables? ¿Qué medidas has llevado a cabo para asegurar la fiabilidad de los resultados?

Para conseguir unos resultados lo más fiables y exactos posibles, todas las tareas se realizaron en el mismo computador y en el periodo de tiempo más corto posible, evitando así que variables como la temperatura o los estados de los servidores afectasen a los resultados. Además, también se tuvo en cuenta que no se realizaban otras tareas en el ordenador que pudieran afectar al rendimiento de este y, consecuentemente a los resultados. Aunque este aspecto es discutible ya que no disponemos de un computador estrictamente para realizar las pruebas y, no podemos asegurar al 100% que el ordenador no estuviese realizando en segundo plano tareas pesadas no controlables por el usuario.

[3] Destaquemos el hecho de que se ha realizado una generalización del tipo de usuario medio. Cada usuario se comporta de manera muy diferente y puede trabajar habitualmente con diferentes tipos de fichero.

7. Fortalezas y debilidades

La principal fortaleza para la estructura del script elegida es su flexibilidad de cara al usuario, permitiéndole elegir en cada ejecución de éste, la tarea a realizar. Puede elegir si crear solo crear la estructura de directorios, eliminando antiguos resultados, si realizar la monitorización des de la localización del archivo que él desee, si realizar los gráficos, combinar dos de los pasos anteriores, o si realizar la ejecución completa del script.

Aunque este grado de flexibilidad puede parecer una gran fortaleza a manos de un usuario medianamente avanzado, deviene su gran debilidad a manos de un usuario inexperto, ya que le permite ejecutar, por ejemplo, las monitorizaciones sin haber creado el árbol de directorios, produciendo así gran cantidad de errores.

Una alternativa de diseño hubiese sido automatizar por completo la ejecución del script, sufrimiento la variable introducida por línea de comandos y realizando cada la ejecución de principio a fin. Empezando por la creación de los directorios, pasando por la monitorización de las cinco transferencias y, finalmente, crear los gráficos de los resultados. Esta solución podría ser muy adecuada para usuarios sin experiencia alguna, aunque, por otro lado, elimine por completo la flexibilidad que tanto es deseada para algunos.

8. Conclusiones

Con la realización de las pruebas desarrolladas y documentadas en la parte superior se pone fin a la práctica donde hemos obtenido una solución al problema con la mayor exactitud posible atendiendo a ciertas limitaciones temporales y económicas.

Hemos realizado la monitorización del sistema durante la realización de las diversas transferencias de un archivo para diferentes niveles de concurrencia.
Una de las problemáticas que hemos afrontado con la realización de las diferentes pruebas ha sido la problemática que presentaban algunos datos incoherentes con los resultados esperados. Hemos realizado diversos análisis y expuesto diversas explicaciones de los motivos de la aparición de dichos datos de carácter dudoso basados en nuestra experiencia a lo largo del curso. Sin embargo este ha sido el factor con el que se ha salido menos satisfecho, ya que el grado de satisfacción ha sido alto a la hora de realizar los diferentes análisis mostrados en la parte anterior del documento.

Uno de los objetivos principales de nuestra práctica fue realizar las diferentes tareas de la manera más eficiente y fácil para el usuario, objetivo que se ha cumplido realizando un script que realiza la automatización de manera eficiente y comprensible para el lector gracias al diseño modular en diferentes scripts englobados por uno solo.
Se ha realizado la totalidad de las tareas en el mismo computador con el objetivo definido en la introducción de obtener los resultados más fiables y exactos posibles.

Con la realización del trabajo se han adquirido capacidades de análisis y monitorización. Junto a esto se han ampliado los conocimientos en manejo del bash de Linux, siendo este parte principal del desarrollo de las capacidades anteriores y de la realización de las diferentes pruebas explicadas a lo largo de este trabajo. Nuestra habilidad a la hora de afrontar errores y problemas diversos y la capacidad para encontrar una solución ha sido incrementada de manera sustancial debido a la alta carga de problemas diversos e inesperados con los que hemos tenido que tratar en la realización de esta práctica.

Hemos llegado a la conclusión de que el tipo de sistema de almacenamiento que será eficiente para el usuario es propio de cada usuario y de sus necesidades. Nos parece una tarea realmente difícil decidir qué tipo de almacenamiento es efectivo para un usuario sin saber sus características. Estas características podrían ser, por ejemplo, el tiempo de uso de los ficheros habituales, el tipo y número de terminales que dispone, la frecuencia con la que realiza el uso de los archivos, etc. No obstante, según el análisis obtenido, y nuestra experiencia, el almacenamiento de archivos en el disco duro siempre es la opción más eficiente, siempre y cuando se tenga el almacenamiento que uno necesita y causas como la disponibilidad de los ficheros en cualquier lugar sean poco valoradas.