Grado en Ingeniería Informática

Evaluación y Comportamiento de los Sistemas Informáticos

Cuaderno de Prácticas

Lluis Barca Pons
lluis.barca1@estudiant.uib.es

6 de junio de 2022

Práctica 1

Para la realización de esta práctica contaremos con los servidores A y B los cuales están dedicados a tareas de cálculo científico. Es decir, las cargas que ejecutan son intensivas en CPU, y, por lo tanto, este es su dispositivo más demandado. A continuación, se detallan las características de cada uno de los servidores.

Servidor A
Nombre del servidor: Dell Power Edge T430 Número de CPUs: 16 Tamaño de la memoria RAM: 7753Mib (≈ 8GB) Coste: 1245€

Servidor B
Nombre del servidor: Dell Power Edge T330 Número de CPUs: 8 Tamaño de la memoria RAM: 15258,8Mib (≈ 16GB) Coste: 907€

El administrador de un centro de datos se enfrenta al reto de decidir qué servidor es más adecuado para la ejecución de una carga intensiva de CPU, el servidor A o el servidor B. Actualmente, el tiempo medio para ejecutar la carga en el servidor es de 31,01 segundos. Para realizar una justa comparación, se ha ejecutado la carga intensiva de CPU en los servidores A y B un total de 10 veces, obteniendo los resultados mostrados a continuación.

	Tiempo de ejecución (s)	
	Servidor A	Servidor B
	24,15	27,01
	23,18	26,18
	25,01	26,56
	23,34	28,02
	22,65	26,78
	24,54	27,43
	23,46	27,34
	22,38	26,04
	23,54	27,19
	23,59	27,43
Tiempo medio:	23,584	26,998

1. ¿Qué servidor resulta más adecuado para el cambio solamente considerando el rendimiento? ¿Por qué? ¿En qué métrica o valor determina la decisión?

Para decidir que servidor es el más adecuado con respecto al actual, tenemos que comprar la aceleración de los respectivos servidores.

$$A : \frac{T_O}{T_A} = \frac{31,01}{23,584} = 1,3149$$

$$B : \frac{T_O}{T_B} = \frac{31,01}{26,998} = 1,1490$$

Finalmente, podemos concluir que, considerando únicamente el rendimiento, el servidor A ofrecería una mayor mejora. Concretamente, el servidor A es aproximadamente 1,3149 veces más rápido, o un 31,49 % mejor, respecto al servidor original.

2. ¿Y si además tenemos en cuenta el coste del servidor? ¿Cuál sería más adecuado? ¿Por qué? ¿En qué métrica o valor te basas?

Si ahora tenemos en cuenta el coste de dichos servidores, deberemos analizarlo con otro índice. En este caso, si le damos más importancia al coste del servidor, obtendremos los siguientes índices de cada uno de ellos. En primer lugar, el índice del servidor A:

$$Ratio - Rendimiento/Coste_A = \frac{1}{Coste_A * T_A} = \frac{1}{1245€ * 23,584s} = 0,000034058 = 3,4058 * 10^{-5}$$

Y, por otro lado, el índice del servidor B:

$$Ratio - Rendimiento/Coste_B = \frac{1}{Coste_B * T_B} = \frac{1}{907€ * 26,998s} = 0,000040838 = 4,0838 * 10^{-5}$$

Ahora, si dividimos los índices

$$\frac{Indice_B}{Indice_A} = \frac{4,0838 * 10^{-5}}{3,4058 * 10^{-5}} = 1,2$$

Podemos observar que el servidor A es un 20 % más rentable que el servidor B.

3. ¿Cómo crees que afectan los recursos hardware de los servidores? ¿Tienen algún tipo de trascendencia en la decisión?

En efecto, los recursos hardware en los servidores son un factor importante a la hora de decidir si un servidor es mejor o peor para el uso que le queramos dar

Habitualmente podemos pensar que la mejora de hardware en un servidor se manifestará como una mejor lineal del rendimiento. Por ejemplo, si tenemos un servidor que ejecuta una carga en un tiempo de ejecución t con una mejora del hardware k, el tiempo de ejecución no será $\frac{t}{2}$ si la mejora realizada es $2k$; esta mejora en función del tiempo no suele seguir una función lineal

En conclusión, la decisión a tomar si tiene algún tipo de transcendencia porque esta mejora de rendimiento se tiene que ver compensada con el coste de dicha mejora.

Práctica 2

Para la realización de esta práctica, el alumno ya contará con el entorno Ubuntu (o el que haya considerado) instalado, siendo totalmente funcional. Para asegurarnos de que el entorno está totalmente listo, se deberá poder acceder al directorio /proc. Además, se contará ya con una herramienta para filtrar y limpiar los ficheros de datos como para realizar representaciones gráficas.

En esta práctica no se tendrá en cuenta qué tipo de actividad está realizando el sistema mientras se realiza la monitorización del mismo. Antes de empezar a responder las diferentes partes, se recomienda probar los monitores, sus filtros, el volcado de ficheros y el tratamiento de los mismos.

Monitorización de la CPU

En esta primera parte, se pide monitorizar la CPU durante 1 hora haciendo uso del monitor TOP. Los datos obtenidos (ÚTILES) deberán ser guardados en un fichero de salida para posteriormente tratarlos y responder a las siguientes preguntas:

---

1. ¿Cuántas CPUs tiene el sistema que se ha monitorizado? ¿De dónde se ha obtenido esa información?

El sistema monitorizado tiene 8 CPUs dedicadas a soportar la carga del trabajo. Esta información se ha obtenido utilizando el comando que se muestra a continuación:

```
luis@luisb:~$ lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes:        Little Endian
CPU(s):                8
```

Donde claramente nos indica que el número de CPUs es 8.

2. ¿Cuál es la utilización media de la CPU en modo usuario, sistema y en global?

Para obtener los datos que se nos pide, se ha monitorizado la CPU durante 1h con intervalos de muestreo de 15 minutos. Se ha decidido tomar este intervalo para poder compararlo con los datos obtenidos de la monitorización de la memoria principal. De esta forma mantendremos una consistencia en los datos presentados.

Concretamente, utilizaremos el monitor TOP para registrar los valores de la CPU del sistema, en modo usuario y la global. El comando concreto es el siguiente:

```
1  top -b -d 15 -n 242 | grep -i "Cpu(s)" > topCPU.txt
```

Podemos observar que en los parámetros le indicamos la frecuencia de muestreo en primer lugar y el total de muestras a realizar, que equivalen a 1h. Observamos un aumento en dos unidades del total de muestras a realizar; esto se debe a que la primera y la última muestra las descartamos para una menor contaminación de los datos (esto se aplicara en todos los procesos de monitorización que se realicen). A continuación, con el comando *grep* indicamos los datos del comando TOP que queremos almacenar; en este caso los de la CPU. Todo ello transferido a un fichero de texto que podremos consultar al final del proceso de monitorización.

Ahora necesitamos sacar la columna que nos interese en cada caso. Para ello utilizaremos el siguiente comando:

```
cat topCPU.txt | awk '{print $x}' > columnaX.txt
```

El comando *cat* nos permite coger el fichero deseado y meterlo en la terminal. A continuación con el comando *awk* podemos filtrar la columna que nos interese. En este caso, filtraremos por columnas, donde x es el número de la columna que queramos almacenar. Concretamente nos interesan las columnas 2, 4 y 6, que son respectivamente la de usuario, de sistema y inactiva. Para obtener la utilización de CPU global se ha restado la inactiva a la total (en este caso 100 %). Finalmente enviamos esta información filtrada a un nuevo fichero para no sobreescribir el de origen.

Con todos estos datos a disposición, podemos realizar las correspondientes medias utilizando una hoja de Excel. Los resultados obtenidos son los siguientes:

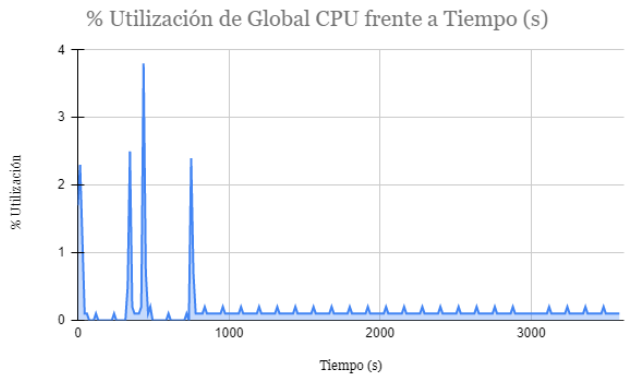
Medias en %		
CPU Global	**CPU Sistema**	**CPU Usuario**
0,17	0,02	0,12

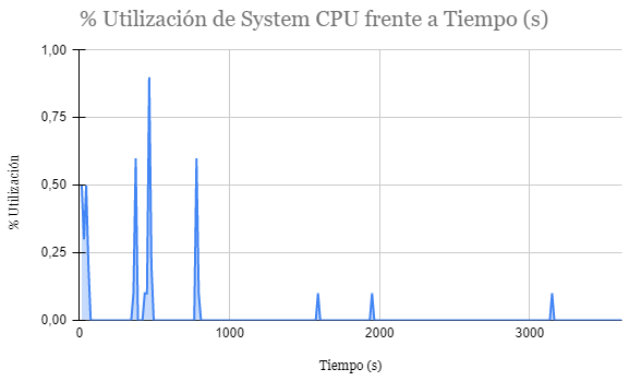
Cabe añadir que este proceso de filtrado por columna se llevara a cabo cuando sea necesario, sin mencionarlo explícitamente, en las próximas monitorizaciones.

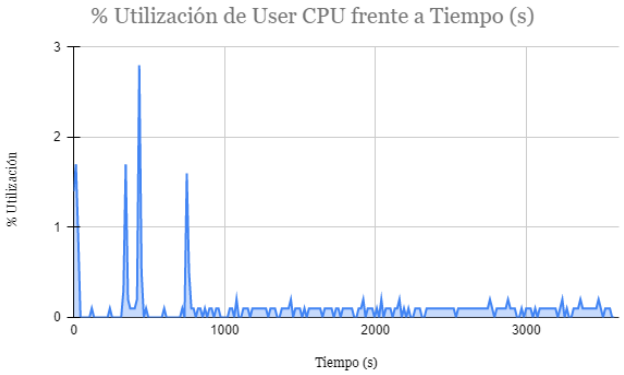
3. ¿Cómo se comportan las medidas anteriores a lo largo del tiempo de observación? Muestra las tres métricas de forma gráfica.

Las métricas expuestas a continuación muestran un comportamiento similar entre ellas. Esto se debe a que en los primeros 15 minutos, se ha realizado algún uso del sistema (aunque sea bajo) y en el resto de tiempo este se encuentra prácticamente inutilizado, ya que no realizaba ningún trabajo.

Es necesario comentar que los ejes Y de las gráficas no tienen el mismo valor máximo. Estos valores son dinámicos con el objetivo de poder observar con mayor detalle las gráficas que tienen valores más pequeños. En cambio, si se mantuviera un valor máximo fijo e igual, alguna detalles de la gráfica no podrian ser apreciados.







4. ¿Cuál es la sobrecarga provocada por el monitor TOP?

La sobrecarga del monitor la obtendremos registrando la información que nos devuelve el monitor TIME cuando ejecutamos el monitor TOP para una única muestra.

```
1  time top -b -d 1 -n 1 | grep -i "Cpu(s)" > sobrecargaTOP.txt
```

Por tanto, obtendremos la sobrecarga dividiendo el tiempo real que tarda el monitor en realizar una muestra (obtenido con el monitor TIME), entre el tiempo de muestreo; como se muestra en la siguiente fórmula:

$$Sobrecarga = \frac{0,182s}{15s} * 100 = 1,21\,\%$$

Monitorización de la memoria principal

En esta segunda parte, se pide monitorizar la memoria principal del sistema durante 1 hora haciendo uso del monitor VMSTAT con un intervalo de muestreo de 15 segundos. Los datos obtenidos (ÚTILES) deberán ser guardados en un fichero de salida para posteriormente tratarlos y responder a las siguientes preguntas:

---

1. ¿Qué capacidad total tiene la memoria principal del sistema? ¿De dónde se ha obtenido ese dato?

Para saber la capacidad total de memoria principal que tiene el sistema, utilizaremos monitor TOP, que entre muchas cosas también nos muestra la memoria total. Esto lo obtendremos con el siguiente comando:

```
top -b -d 1 -n 1 | grep -i "KiB Mem"
```

Al monitor TOP le asignamos unos parámetros, simplemente para que no este activo constantemente, y a continuación con el comando *grep* podemos extraer el valor de la memoria principal; el cual nos dará: **8074700 KiB**.

2. ¿Cuál es la utilización media de la memoria? ¿Y la capacidad media utilizada?

Para obtener los datos para realizar las respectivas medias, se ha monitorizado la memoria principal durante 1h en un intervalo de 15 segundos. Concretamente, utilizaremos el monitor VMSTAT para registrar los valores de la memoria libre. El comando concreto es el siguiente:

```
vmstat 15 242 -n > vmstatMEM.txt
```

Una vez recogidos todos los datos, podemos obtener la columna que nos informa del valor de la memoria libre del sistema realizando el siguiente comando, para efectuar el filtrado:

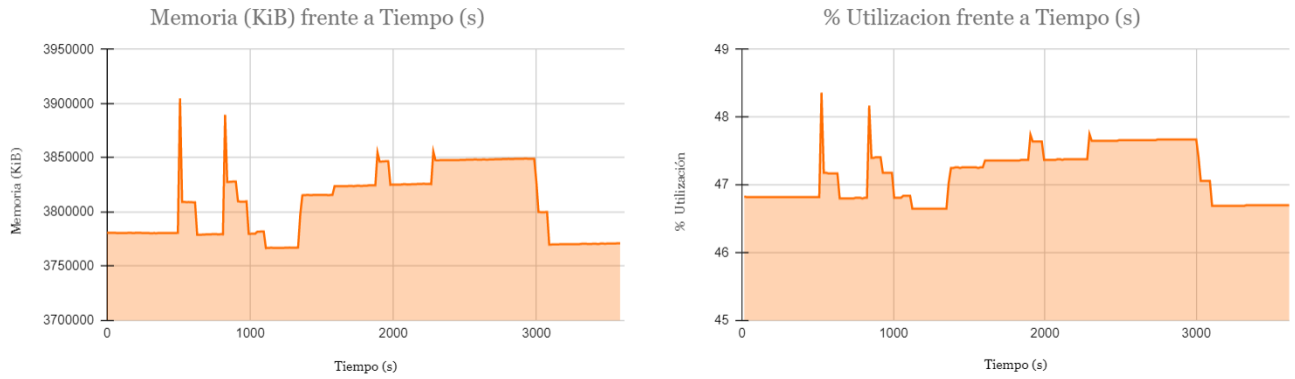
```
cat vmstatMEM.txt | awk '{print $4}' > MEMlibre.txt
```

Finalmente, en este último fichero tendremos todos los valores de memoria libre que ha registrado el monitor. Para obtener la memoria utilizada, debemos restar al total de la memoria principal el valor de la memoria libre. Es con este valor con el que se ha calculado tanto la utilización media como la capacidad media utilizada de la memoria, utilizando una hoja de Excel. Los resultados obtenidos son los siguientes:

Utilización Media (KiB)	Capacidad Media (%)
3.806.916	47,15

3. ¿Cómo se comporta la utilización de la memoria y la capacidad utilizada? Representa estas métricas gráficamente.

Durante el proceso de monitorización, observamos dos picos en las gráficas. Estos se deben a que en los primeros minutos del proceso de monitorización, estaban abiertas ciertas ventanas del gestor de ficheros; además de realizarse algunos movimientos de ficheros entre directorios. El resto del tiempo no se realizó ningún trabajo concreto, por tanto, la carga de trabajo que vemos se deberá a los procesos internos del propio sistema operativo.



4. ¿Cuál es la sobrecarga provocada por el monitor VMSTAT?

Para obtener la sobrecarga del monitor, primero deberemos registrar el tiempo de ejecución del monitor VMSTAT; para luego dividirlo entre el tiempo de muestreo. Para lo comentado en primera instancia, utilizaremos el monitor TIME, el cual nos permitirá registrar el tiempo de ejecución del monitor TOP para una única muestra.

```
time vmstat 1 1 -n > sobrecarga_VMSTAT.txt
```

Por tanto, obtendremos la sobrecarga como se muestra en la siguiente fórmula:

$$Sobrecarga = \frac{0,007s}{15s} * 100 = 0,05\,\%$$

Monitorización de la CPU y de la memoria principal al mismo tiempo

Se desea monitorizar un sistema informático con el fin de conocer el comportamiento de la CPU y la memoria principal. De la CPU se desea estudiar la utilización en modo usuario, sistema y la utilización de forma global. Por otra parte, de la memoria principal se desea estudiar la capacidad y la utilización de la misma.

---

1. Determinar el monitor o monitores más adecuados para obtener la información solicitada.

Los monitores más adecuados para obtener dicha información son, por un lado, el TOP para monitorizar los valores de la CPU y el VMSTAT para la memoria principal.

2. Monitorizar el sistema durante 90 minutos, recogiendo la información útil cada 30 segundos. La información útil deberá ser almacenada en un fichero de salida.

Para monitorizar el sistema se utilizarán los monitores comentados en el apartado anterior, ambos ejecutados a la vez, en terminales diferentes. Por un lado, el monitor TOP:

```
1  top -b -d 30 -n 182 | grep -i "Cpu(s)" > topCPU.txt
```

Sus parámetros nos indica que la frecuencia de muestreo será cada 30 segundos y en total se realizaran 182 muestras. Todo ello almacenado en un fichero de texto de salida, para su posterior filtrado y tratado de datos.

Esto equivale a los 90 minutos exigidos en el enunciado. Aunque como siempre, realizamos dos muestras extras para descartar los extremos, con la finalidad de tener unos datos menos contaminados.

Y, por otro lado, el monitor VMSTAT:

```
1  vmstat 30 182 -n > vmstatMEM.txt
```

El cual también vemos una semejanza con el monitor VMSTAT en cuanto a los parámetros establecidos. Toda la información recogida, al igual que con el otro monitor, se almacenara en un fichero de texto de salida para su posterior filtrado y tratado.

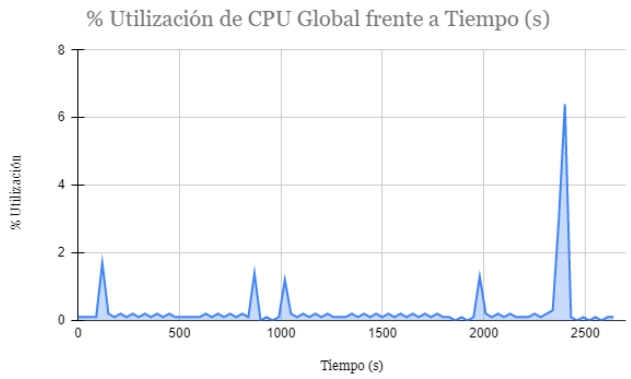
3. Calcular la media para la utilización de la CPU en modo usuario, sistema y la utilización global. También, calcular la media para la capacidad y la utilización de la memoria principal.

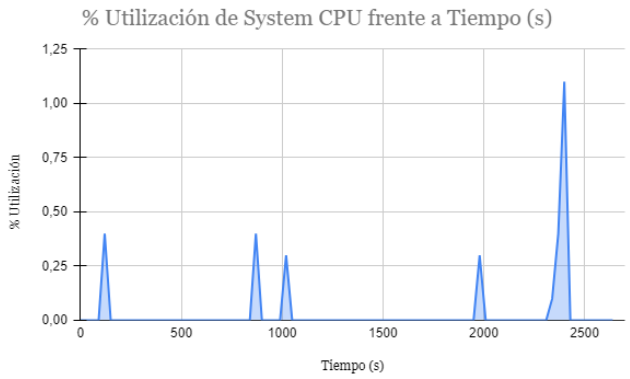
En este apartado, con todo el trabajo realizado en los apartados anteriores, solo tendremos que introducir los datos en un Excel para calcular las medias. Los resultados son los que se muestran a continuación:

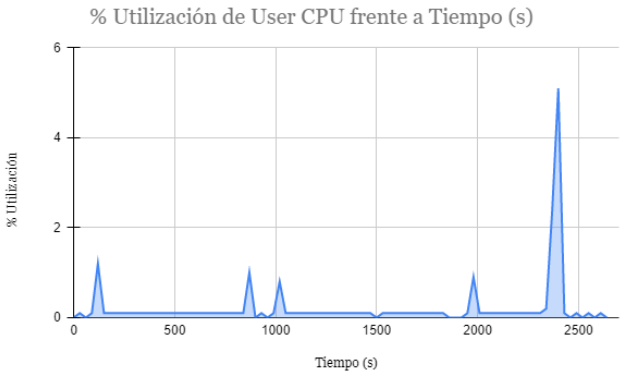
Medias				
CPU			Memoria Principal	
Global	Sistema	Usuario	Capacidad	Utilización
0,28 %	0,03 %	0,21 %	3.953.994 KiB	48,97 %

4. Graficar todos los resultados obtenidos durante los 90 minutos de monitorización del sistema.

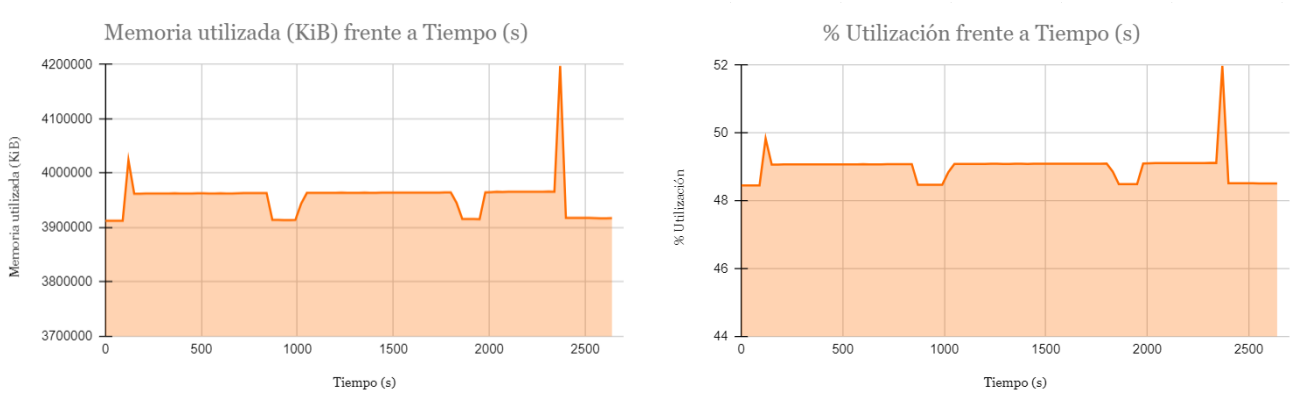
A continuación los datos graficados de la CPU:







Y aquí los datos de la memoria principal:



5. Calcular la sobrecarga que ejerce el monitor (o los monitores) sobre el sistema.

La sobrecarga en este caso será la de los dos monitores a la vez. Esto es así porque tenemos dos monitores ejecutándose en paralelo, que consumirán recursos a la vez. Por tanto, la sobrecarga final que ejercerán estos monitores sobre el sistema será la suma de ambos monitores.

Con esta imagen se intenta mostrar de forma gráfica el paralelismo de los monitores. Es por eso que debemos realizar la suma de ambos, para obtener la sobrecarga total sobre el sistema.

Primero calcularemos la sobrecarga que ejerce el monitor TOP:

$$Sobrecarga = \frac{0,182s}{30s} * 100 = 0,6\,\%$$

Y ahora la sobrecarga que ejerce el monitor VMSTAT:

$$Sobrecarga = \frac{0,007s}{30s} * 100 = 0,02\,\%$$

Finalmente, nos queda esta sobrecarga:

$$Sobrecarga = Sobrecarga_{TOP} + Sobrecarga_{VMSTAT} = 0,6\,\% + 0,02\,\% = 0,62\,\%$$

Práctica 3

En la práctica 1 se presentaron los servidores A y B como alternativas al sistema actual. Además, se afirmó que los dos servidores ejecutaban cargas de CPU, concretamente, el servidor A ejecuta la carga Sysbench CPU y el servidor B la carga Stress-ng. A continuación, se muestran los tiempos de ejecución de cada servidor para determinadas configuraciones de carga y % de uso de la CPU:

	Tiempos de ejecución (s)	
	Servidor A	**Servidor B**
Configuración	Carga = 250000 % CPU = 50	Carga = 300000 % CPU = 100
Ejecución 1	24,15	27,01
Ejecución 2	23,18	26,18
Ejecución 3	25,01	26,56
Ejecución 4	23,34	28,02
Ejecución 5	22,65	26,78
Ejecución 6	24,54	27,43
Ejecución 7	23,46	27,34
Ejecución 8	22,38	26,04
Ejecución 9	23,54	27,19
Ejecución 10	23,59	27,43
Tiempo medio	23,584	26,998

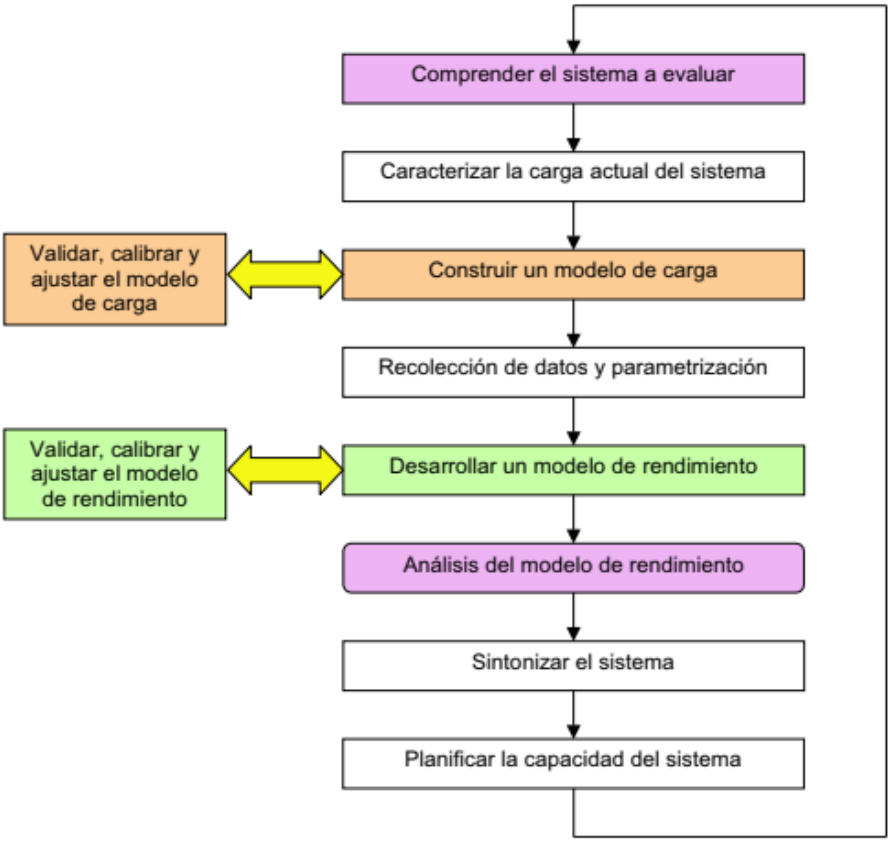
Evaluación del sistema actual

Para la evaluación del sistema actual, se utilizará la carga Sysbench CPU con un porcentaje de uso de la CPU del 50 %, la cual se ejecutará en el sistema actual y se harán uso de las técnicas de monitorización ya aprendidas en la práctica anterior. De este modo, se pide responder a las siguientes preguntas.

---

1. Explica con detalle cómo es el diseño y la implementación del experimento para evaluar el sistema actual. Se deben justificar las decisiones tomadas, desde el número de muestras que se van a tomar hasta qué monitores se van a lazar y por qué.

Para llevar a cabo el diseño y la implementación de este experimento, se seguirá el siguiente diagrama que representa una metodología para el estudio del rendimiento de sistemas.



En primer lugar, el sistema que vamos a evaluar tiene las siguientes propiedades; todas ellas obtenidas a través del comando *lscpu* u obtenidas de prácticas anteriores:

Propiedas Sistema Actual	
Arquitectura	64 bits
Capacidad de memoria	8074700 KiB
Procesador	Intel Core i5-8250U 1.6-3.4GHz
Número de CPUs	8

A continuación definimos las cargas a ejecutar en el sistema actual. Estas serán lanzadas con el benchmark *sysbench* (centrado en la obtención de números primos) y serán de 25000, 50000, 100000, 150000 y 250000. Las cuatro primeras se ejecutarán porque básicamente es lo que se nos pide en el enunciado y la última será para, en los siguientes apartados, poder comparar el sistema actual con uno de los propuestos; que se ha estresado con el mismo benchmark y con una carga de 250000. También se configurara el benchmark para que estrese el 50 % de la CPU, lo que quiere decir que estresara un total de 4 CPUs.

Con este conjunto de cargas podremos observar si al aumentar el número de números primos a buscar, el tiempo de respuesta del sistema aumenta de forma lineal o no. Dependiendo de dichos resultados, podremos afirmar que no hay la misma cantidad de números primos entre los distintos rangos, a pesar de ser una carga el doble que la otra.

Para realizar el experimento, se utilizará el *sysbench* (como se ha comentado anteriormente) para estresar la CPU y los monitores que recogerán las muestras necesarias son el *top*, para la CPU y *vmstat* para la memoria. Para calcular la frecuencia de muestreo de los monitores, se ha ejecutado un *sysbench* para cada carga mencionada anteriormente, para así obtener el tiempo que tarda cada una. A continuación esta carga se ejecuta 5 veces para posteriormente hacer una media del tiempo de respuesta resultante. Esto se realiza de esta forma para garantizar, estadísticamente, que esos tiempos de respuesta son realmente los que tardan cada carga (aunque para un estudio real se deberían hacer 10 o más ejecuciones). Se ha decidido la frecuencia de muestreo en función del tiempo de respuesta más bajo; en este caso el de la primera carga. A continuación los diferentes tiempos de respuesta:

Carga	**Tiempo de respuesta (s)**	**Ejecuciones**	**Total (s)**
25000	8 + 2	5	50
50000	20 + 2	5	110
100000	55 + 2	5	285
150000	100 + 2	5	510
250000	202 + 2	5	1020
		Suma total:	1975

Podemos observar un tiempo añadido en cada carga de 2 segundos. Este es porque realizamos una monitorización (tanto de la CPU como de la memoria) y en el script de monitorización tenemos un *sleep* de 2 segundos cada vez que acaba una ejecución de cualquier carga. Con esto conseguimos

saber cuando inicia y finaliza una ejecución de una carga por los picos que se producen en las gráficas. Por tanto, para saber cuantas veces deberán registrar los monitores el total de muestras, se calcula dividiendo el tiempo total de respuesta entre la frecuencia de muestreo.

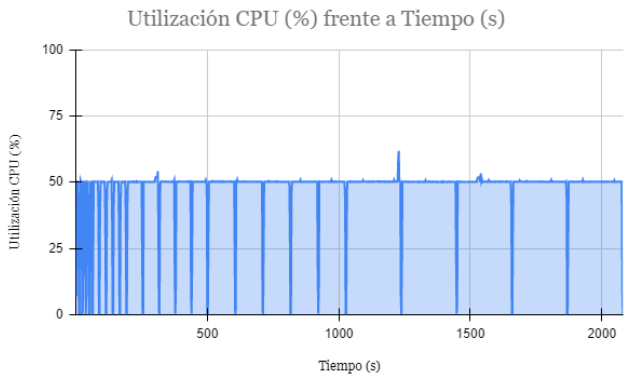
$$Muestras = \frac{1975}{2} = 988$$

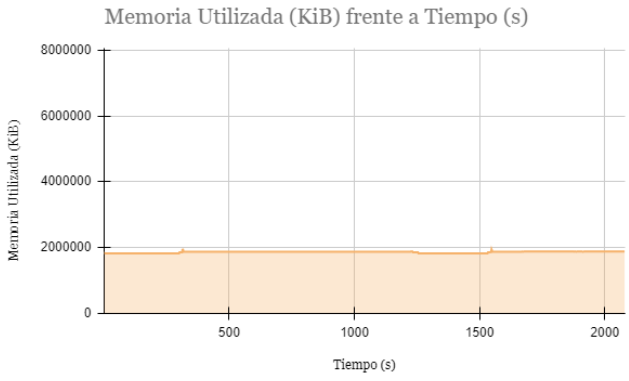
De todas formas, para realizar la monitorización real, se han cogido algunas muestras extras por si un caso algún monitor tuviera algún tipo de problema o para corregir el redondeo de los tiempos de respuesta de cada carga obtenidos anteriormente. Una vez obtenidos todos estos parámetros, podremos ejecutar los respectivos scripts que nos realizaran el trabajo de forma secuencial y automática.

Cabe mencionar que las condiciones del sistema actual han intentado ser las más óptimas posibles. Se ha mantenido el sistema con un 100 % de carga, enchufado a la corriente eléctrica y a temperatura ambiente, durante todo el transcurso del experimento. Además, no se ha utilizado el sistema para realizar ninguna otra tarea que no fuese ejecutar el respectivo benchmark o los monitores. Únicamente podía haber procesos propios del sistema operativo de autogestión en los cuales el usuario no tiene acceso. Todo ello con el objetivo de conseguir unas muestras fiables y aptas para poder comparar posteriormente con otros sistemas de la forma más igualada posible.

2. ¿Cómo se comporta el sistema actual si variamos la carga varía en 25000, 50000, 100000 y 150000 números primos? ¿Cómo es el comportamiento del tiempo de respuesta y la productividad? Indica el valor para cada una de las ejecuciones del experimento y la media entre todos ellas.

Si variamos la carga entre los valores expuestos, el sistema se va a comportar de la misma forma, como vemos en las gráficas generales a continuación:





En la primera observamos la utilización de CPU que se mantiene al 50 %, como era de esperar. Al lanzar un benchmark que estresa este componente al 50 %, lo esperado es que durante todas las cargas el tanto por ciento de CPU sea ese. Cabe mencionar que se nos pide observar como

se comporta el sistema en cada carga, pero en las gráficas se muestra el conjunto de ellas. El objetivo es poder visualizar con mayor perspectiva y no mostrar tantas representaciones gráficas que podrían dificultar el entendimiento de los datos recogidos. Por tanto, con esta representación podemos apreciar las diferentes ejecuciones y/o cargas y sus respectivos valles.

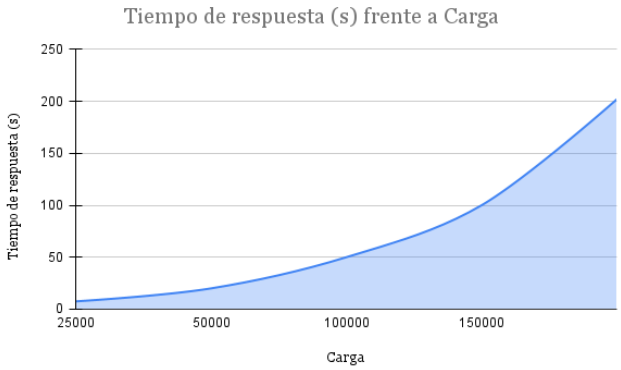
Por otro lado, por lo que hace a la memoria, también se ha representado con una perspectiva global de todas las cargas. Apreciamos que dicha memoria se mantiene prácticamente igual durante el transcurso del experimento. De hecho, era lo esperado, ya que las cargas inyectadas (búsquedas de números primos) no intervienen en dicha memoria y, por tanto, su uso solo se debe a lo que el propio sistema necesita para estar funcionando.

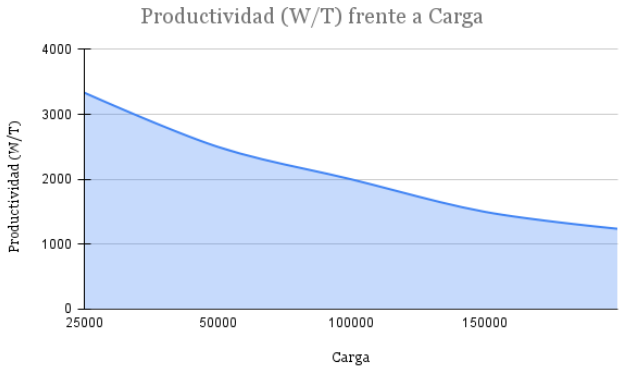
A continuación los valores obtenidos de cada carga tanto de los tiempos de respuesta como de la productividad:

Carga	Tiempo de respuesta (s)	Productividad (W/T)
25000	8	3333
50000	20	2500
100000	55	1818
150000	100	1500
250000	202	1238

Medias	77	1822

Seguidamente los gráficos respectivos:

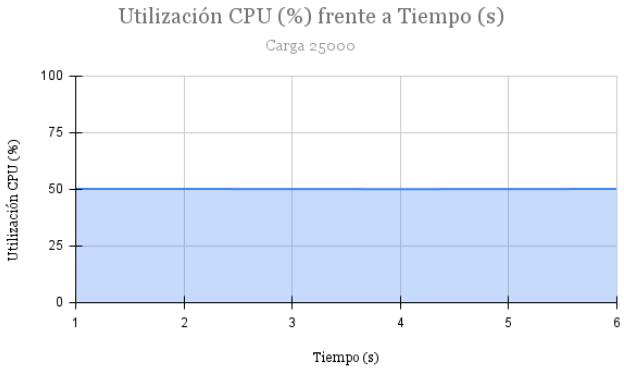


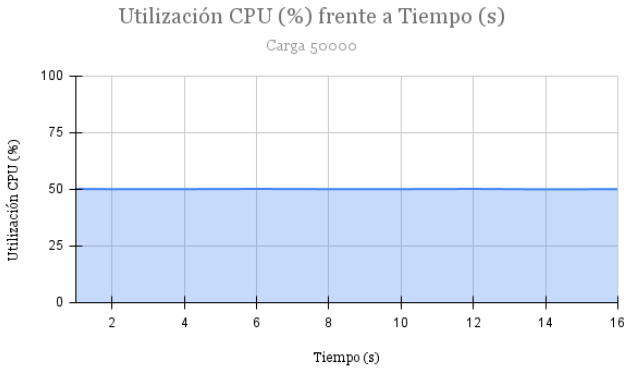


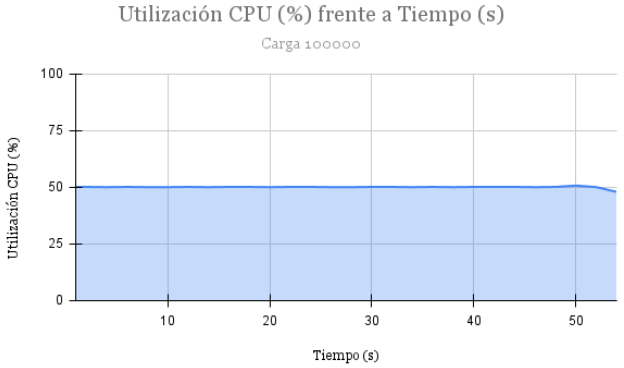
El tiempo de respuesta, como podemos observar en los valores obtenidos, tiende subir de forma bastante pronunciada. Por otro lado, la productividad tiende a bajar a medida que la carga aumenta. Esto nos indica que a un mayor número de cargas, el sistema tiende a tardar más y a ser menos productivo; pero no sigue una tendencia lineal. Esto se debe realmente a que cada vez nos encontramos con un menor número de números primos. Además de que al no tener "memoria", cada vez que aumentamos la carga, el sistema debe recorrer los números primos que ya ha recorrido.

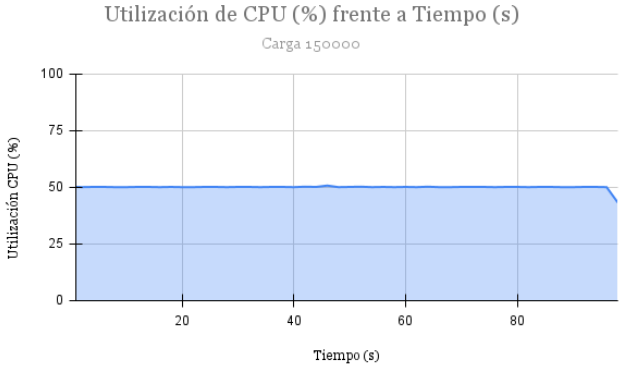
3. ¿Cuál es el porcentaje de CPU y de memoria del sistema para cada una de las cargas ejecutadas? ¿Por qué se produce ese comportamiento? Muéstralo gráficamente a lo largo del tiempo de ejecución de la carga.

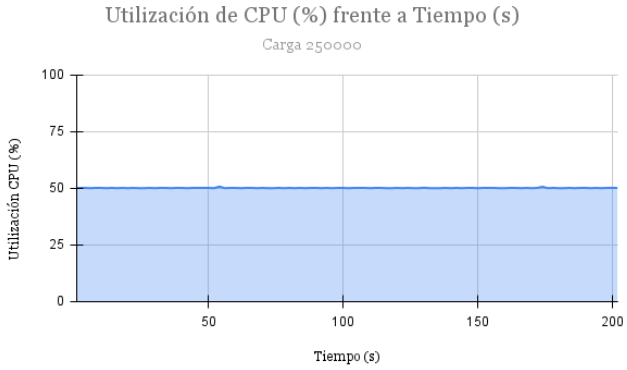
A continuación las respectivas gráficas de la utilización de CPU de cada carga:



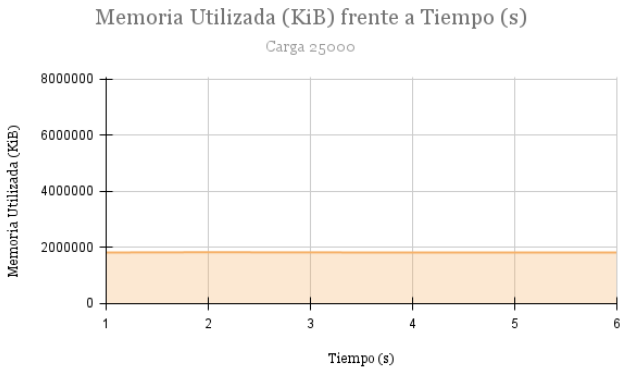


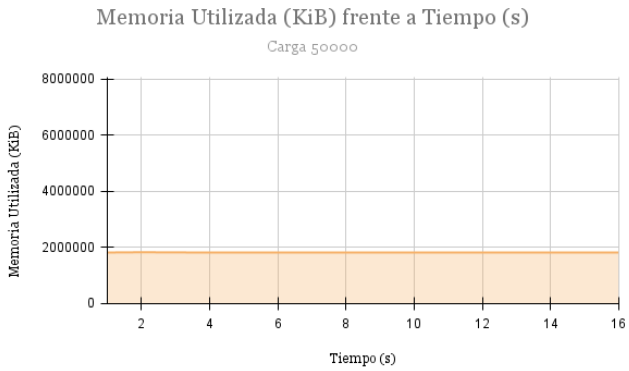


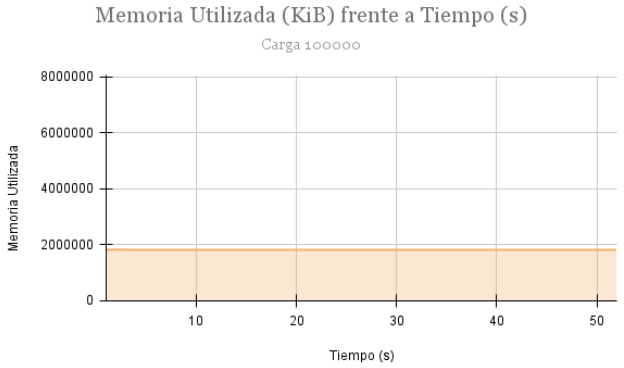


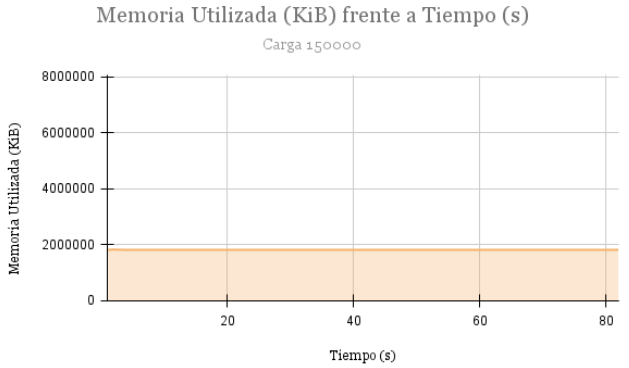


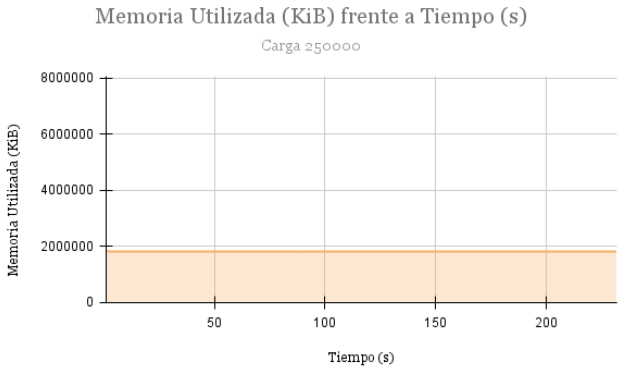
Seguidamente, las gráficas de la capacidad de memoria utilizada:











Como observamos, tanto en la CPU como en la memoria, los valores se mantienen igual. Esto, como se ha comentado durante el experimento, es lo esperado; ya que el *sysbench* estresa el 50 % de la CPU y la memoria no se utiliza para realizar la búsqueda de números primos.

Comparación del sistema actual con el servidor A y B

Para la comparación del sistema actual con los servidores A y B se harán uso de los datos proporcionados anteriormente en la tabla. Se deberá aplicar la técnica de benchmarking para realizar la comparación teniendo en cuenta la carga que ejecuta cada uno de los servidores, así como su nivel de uso de los recursos de CPU. Además, se deberán contestar las siguientes cuestiones.

---

1. Explica detalladamente cómo se va a realizar el proceso de benchmarking. Se deberán justificar todas las decisiones que se han tomado al respecto.

Para llevar a cabo el diseño y la implementación de este experimento, se seguirá el diagrama expuesto en el apartado anterior.

El sistema actual tiene las propiedades representadas en la tabla del apartado anterior y la carga ejecutada con el benchmark *stress* será de 300000. Esta carga se ejecutará 5 veces para garantizar que los datos obtenidos son lo más realista posibles. Para ello, se utilizarán los scripts mencionados al principio del experimento y que se podrán encontrar al final de esta documentación. En este caso, el benchmark *stress* será configurado para estresar el 100 % de las CPUs, que corresponde a un total de 8.

Por lo que hace a los monitores, se utilizaran los mismos que en el apartado anterior, el *top* para monitorizar el estado de la CPU y el *vmstat* para hacer lo respectivo con la memoria. Estos monitores tienen como parámetros la frecuencia de muestreo y el total de ejecuciones que debería hacer el monitor. Como la carga tarda aproximadamente 238s en ejecutarse por completo, ejecutar 5 veces dicha carga más un *sleep* de 2s entre ejecuciones, nos deja un tiempo total de 1200s; lo cual equivale a 600 muestras. De todas formas, como es habitual, recogeremos algunas muestras más con el monitor para corregir el redondeo de los tiempos de respuesta obtenidos.

Por lo que hace a las condiciones del sistema actual, se han buscado las condiciones óptimas para realizar el experimento con la mayor objetividad posible. Se ha mantenido el sistema a un 100 % de carga y conectado a la red eléctrica durante toda la prueba. Además, no se ha utilizado el sistema para realizar ninguna otra tarea que no fuera ejecutar el benchmark y los monitores. Únicamente se han podido ejecutar procesos propios del sistema operativo, necesarios para mantenerlo activo.

2. Considerando solamente el rendimiento de los servidores, ¿cambiarías el sistema actual por el servidor A? ¿Y por el B? ¿Por qué?

Si tenemos en cuenta únicamente el rendimiento de ambos sistemas, entonces deberemos comparar las respectivas productividades:

$$X_A = \frac{Trabajo}{Tiempo} = \frac{250000W}{23,584s} = 10600W/s$$

$$X_B = \frac{Trabajo}{Tiempo} = \frac{300000W}{26,998s} = 11112W/s$$

$$X_{Actual-Sysbench} = \frac{Trabajo}{Tiempo} = \frac{250000W}{202s} = 1237W/s$$

$$X_{Actual-Stress} = \frac{Trabajo}{Tiempo} = \frac{300000W}{237,41s} = 1263W/s$$

Observamos claramente como ambos servidores son mejores que el sistema actual. Cabe recordar que no sería una comparación justa decir que el Servidor B tiene una mejor productividad que el Servidor A, ya que ambos han sido estresados con benchmarks distintos, totalmente diferentes.

Si ahora nos disponemos a hacer las respectivas aceleraciones:

$$A_{A-Actual} = \frac{X_A}{X_{Actual-Sysbench}} = \frac{10600}{1237} = 8,57$$

$$A_{B-Actual} = \frac{X_B}{X_{Actual-Stress}} = \frac{11112}{1263} = 8,8$$

Entonces podemos afirmar que ambos servidores son, aproximadamente, 8 veces mejor que el sistema actual. Y, por tanto, realizar el cambio del sistema actual por ambos servidores sería una decisión acertada.

3. Si además consideramos el coste económico, ¿cambiarías el sistema actual por el servidor A? ¿Y por el B? ¿Por qué?

En primer lugar, debemos calcular el rendimiento que hemos sacado sobre el dinero invertido en cada sistema. Esto se calculará dividiendo la productividad entre el coste de cada uno.

$$Rendimiento - Coste_A = \frac{X_A}{Coste_A} = \frac{10600}{1245€} = 8,51$$

$$Rendimiento - Coste_B = \frac{X_B}{Coste_B} = \frac{11112}{907€} = 12,25$$

$$Rendimiento - Coste_{Actual-Sysbench} = \frac{X_{Actual}}{Coste_{Actual}} = \frac{1237}{700€} = 1,77$$

$$Rendimiento - Coste_{Actual-Stress} = \frac{X_{Acual}}{Coste_{Actual}} = \frac{1263}{700€} = 1,8$$

Aunque la forma habitual de calcular el rendimiento coste sea:

$$X = \frac{1}{Rendimiento \times Coste}$$

creo que calculando las productividades, el índice resultante es más visual. De todas formas, ambos cálculos son linealmente y darían resultados proporcionales.

Si ahora nos disponemos a calcular las respectivas aceleraciones:

$$A_{A-Actual} = \frac{Rendimiento - Coste_A}{Rendimiento - Coste_{Actual-Sysbench}} = \frac{8,51}{1,77} = 4,8$$

$$A_{B-Actual} = \frac{Rendimiento - Coste_B}{Rendimiento - Coste_{Actual-Stress}} = \frac{12,25}{1,8} = 6,8$$

Por tanto, podemos observar que el servidor A es 4,8 y el servidor B 6,8 veces mejor en lo que se refiere al rendimiento sacado sobre la inversión realizada, que el sistema actual. Por tanto, realizar un cambio de cualquiera de los servidores por el sistema actual, sería una decisión totalmente acertada.

4. ¿Cómo influye la carga ejecutada en los servidores A y B en la decisión de cambio?

En primer lugar, hay que diferenciar entre los benmchmarks ejecutados en cada servidor. Por un lado, el *sysbench* realiza una búsqueda de números primos entre 0 y la carga inyectada (por ejemplo 25000). Y en este caso, al no haber una proporción entre el número de números primos que hay entre dos valores, a mayor carga el tiempo de respuesta no aumenta linealmente. Por otro lado, el *stress* realiza diferentes operaciones con enteros, números en coma flotante, pruebas del registro 0 del procesador, entre otras. La gran diferencia es que dichas operaciones si muestran una tendencia lineal cuando la carga varia.

En conclusión, la carga ejecutada en los servidores si influye en la decisión de cambio; todo dependerá del objetivo con el que queramos adquirir un sistema. Si comparamos, respectivamente, cada servidor con el sistema actual, entonces podemos afirmar con total seguridad que los servidores son mejores. Pero no podemos asegurar que el servidor B sea mejor o peor que el A realizando búsquedas de números primos, o viceversa con los diferentes tipos de operaciones.

Scripts

A continuación el script que ejecuta los distintos monitores:

```
#!/bin/bash

if [ -z "$1" ]; then # Check monitor
    echo "Define a valid moitor <top, vm>"
    exit 1
else
    case $1 in
            help)
                echo "Usage: mon <top vm> <delay> <messures>"
                exit 1
                ;;
            top);;
            vm) ;;
            *)
            echo "Define a valid moitor <top, vm>"
            exit 1
                ;;
    esac
fi

if [ -z "$2" ]; then # Check delay
    echo "Define a delay"
    exit 1
else

    if [ "$2" -lt 1 ]; then # delay < 1
        echo "Define a valid range of delay. d > 1"
        exit 1
    fi
fi

if [ -z "$3" ]; then # Check messures
    echo "Define the number of messures"
    exit 1
else
    if [ "$3" -lt 1 ]; then # messures < 1
        echo "Define a valid range of messures. n > 1"
        exit 1
    fi
fi
```

```
41
42  # Correct syntax
43  mkdir "./monitor" > /dev/null 2>&1
44  echo "=== Starting monitor ==="
45
46  printf "Work in process... "
47  case $1 in
48          top)
49                  top -b -d "$2" -n "$3" | grep "%Cpu(s):" > tempCPU
50
51              sed -i 's/ni,100.0/ni, 100.0/g' tempCPU    # Add space to nice
                ↪  time. Avoid bug with awk
52          awk '{print $2, $4, $8}' tempCPU > "./monitor/TOP.txt"  # Data
            ↪  filtering
53          rm tempCPU
54          ;; #break
55
56      vm)
57          vmstat "$2" "$3" -n >tempVM
58
59          sed -i '1,2d' tempVM    # Remove first and second line (header)
60          awk '{print $4}' tempVM > "./monitor/VM.txt"
61          rm tempVM
62          ;; #break
63  esac
64
65  echo "Done"
```

A continuación el script que ejecuta los distintos benchmarks:

```
#!/bin/bash

case $1 in
        help)
                echo "Usage: bench <sysbench stress> <number cpus> <number
                ↪  iterations>"
                exit 1;
                ;;
        sysbench) ;;
        stress) ;;
        *)
                echo "Define a valid monitor <sysbench stress>"
                exit 1
                ;;
esac

if [ -z "$2" ]
then # Check number of cpus
        echo "Define the number of cpus"
        exit 1
else
        if [ "$2" -lt 1 ]
        then # Number of cpus < 1
                echo "Define a valid number of cpus"
                exit 1
        fi
fi

if [ -z "$3" ]
then # Check number of iterations
        echo "Define the number of iterations"
        exit 1
else
        if [ "$3" -lt 1 ]
        then # Number of iterations < 1
                echo "Define a valid number of iterations n > 1"
                exit 1
        fi
fi


```

```
for i in 25000 50000 100000 150000 250000
do
        echo "=== Start workload $i ==="
        j=1

        while [ "$j" -le "$3" ]; do
                echo "$1 iteration $j..."

                case $1 in
                        sysbench)
                                sysbench --test=cpu --cpu-max-prime="$i"
                                ↪  --num-threads="$2" run
                                ;; # Break

                        stress)
                                stress-ng --cpu="$2" --cpu-ops="$i"
                                ;; # Break
                esac

                echo "Done"
                j=$((j + 1))

                sleep 2
        done
        echo ""
done
```

* Cabe remarcar que en caso de querer usar otras cargas, se deberán modificar desde el script.

Práctica 4

Bloque 1

Se pide responder las siguientes cuestiones:

---

1. ¿Cuál es la principal diferencia entre la evaluación de sistemas mediante la experimentación real y el modelado?

La principal diferencia es que con la primera estamos trabajando con un sistema real y con la segunda con uno "inventado". En consecuencia, el sistema real no sigue ninguna función (ya que el sistema puede experimentar diferentes actividades no esperadas) y el modelo si, es por eso que nunca va a fallar (independientement del sistema real).

2. ¿Qué relación hay entre ellas?

La relación es que podemos trabajar sobre ambas para realizar los cálculos necesarios para la implementación de un sistema cualquiera. Es decir, el modelo nos podria servir para inyectar posibles cargas reales y poder evaluar como se comporta el sistema. Por otro lado, la experimentación real ayuda a mejorar el modelo que tenemos.

3. ¿Cómo podríamos combinar ambas formas de evaluación? Explica detalladamente cómo podríamos combinarlas con un ejemplo de la vida real.

Ambas formas deberían combinarse. Básicamente, porque el objetivo de los modelos es recrear de manera ficticia la realidad; con el objetivo de estresar al sistema con ese modelo y observar si soportaría una situación real similar. Aunque debemos ser conscientes de que el modelo es algo inventado y, por tanto, la realidad siempre puede sorprendernos. En consecuencia, tener un modelo es necesario para predecir (en la medida de lo posible) lo que nuestro sistema deberá soportar.

Para entenderlo mejor, el ejemplo del servidor es muy claro. Tenemos un servidor que anualmente recibe una serie de peticiones (por ejemplo registros de matrículas de una universidad) que suelen ser similares, año tras año. Entonces, el administrador del sistema puede recrear un modelo con los datos obtenidos del mismo sistema (o datos externos, como por ejemplo el número de personas que ha aprobado selectividad en la misma zona) y testear el servidor para ver si soportara la carga. A partir de allí ya seria decisión del administrador si los resultados obtenidos son fiables o no para afrontar el siguiente periodo de matrículas.

Bloque 2

Consideremos la carga Stress-ng ejecutada en la práctica 3. En concreto, el que es ejecutado al 100 % de uso de la CPU. De la ejecución de esta carga se obtuvo un tiempo medio de respuesta. Además, ahora se concreta que el tiempo medio de ejecución es de 1,23 segundos. También, la CPU recibe una media de dos programas por segundo (del tipo Stress-ng).

Se pide calcular:

---

1. Utilización media del procesador.

Si aplicamos la hipótesis de equilibrio de flujo, entonces podemos aplicar la Ley de la Utilización. En este caso, al recibir la CPU 2 trabajos por segundo, nuestra $\lambda = 2$. Además, nuestro tiempo medio de ejecución es de 1,23s, como se indica en el enunciado. Por tanto, la utilización se calcularía de la siguiente forma:

$$U = X \times S = \lambda \times S = 2 \times 1,23 = 2,46 \simeq 246\,\%$$

Como la utilización es mayor al 100 % entonces podemos deducir que el sistema está saturado y, por tanto, es seguro que si no ha colapsado, colapsara en cualquier momento. Entonces, para que el sistema tenga una utilización del 100 %, necesitaríamos reducir el tiempo de servicio a 0,5s o bajar la productividad a 0,813 trabajos/s.

2. Tiempo medio de espera en la cola del procesador.

El tiempo de espera, es aquel en el cual un trabajo se encuentra en la cola de trabajos para ser servido. Este se puede calcular si restamos al tiempo total, el tiempo de ejecución. Por tanto, el resultado se calcularía de la siguiente forma:

$$W_i = R_i - S_i = 26,998s - 1,23s = 25,768s$$

3. Número medio de programas en la cola de espera del procesador.

Para calcular el número medio de trabajo en la cola de espera, podemos aplicar la Ley de Little a la misma cola. La productividad del procesador es la misma que la de la cola y el tiempo de respuesta es el tiempo de espera en cola calculador en el apartado anterior. Por tanto, se calcula de la siguiente forma:

$$N = X \times R = 2 trabajos/s \times 25,768s = 51,536 trabajos$$

Práctica 5

Bloque 1

Esta práctica consiste en implementar y verificar el correcto funcionamiento del algoritmo del valor medio para resolver redes cerradas (MVA). El algoritmo se puede entrar en la página 136 del libro de la asignatura. También, se pueden hacer uso de los ejercicios resueltos en el libro para comprobar que el algoritmo funciona correctamente.

Además, se deberán mostrar gráficamente el valor de las variables R_i, R, X, N_i y U_i en función del número de clientes que hay en el sistema. Se deberá entregar:

---

1. Implementación del algoritmo en el lenguaje que el alumno considere más oportuno.

El código implementado se ha adjuntado al final de esta práctica. Es un script en Python que calcula el algoritmo del valor medio para resolver redes cerradas y además, genera los gráficos correspondientes.

2. Un documento donde se reflejen tres ejemplos usados para verificar el buen funcionamiento del algoritmo.

Se han realizado tres experimentos para verificar el buen funcionamiento del algoritmo. A continuación se muestran las condiciones y los resultados de los mismos.

Experimento 1

En este experimento se han introducido un tiempo de reflexión de 5 segundos y un total de 5 trabajos. Además, las razones de visita y los tiempos de servicio de cada dispositivo son 15 y 0,03s para el dispositivo 0 y 14 y 0,5s para el dispositivo 1, respectivamente.

Trabajos	R (s)	X (trabajos/s)
1	7.45	0.0803
2	11.402	0.1219
3	16.8098	0.1376
4	23.2072	0.1418
5	30.0414	0.1427

Trabajos	Dispositivo	Ri (s)	Xi (trabajos/s)	Ni (trabajos)	Ui (%)
1	0	0.03	1.2048	0.0361	0.0361
	1	0.5	1.1245	0.5622	0.5622
2	0	0.0311	1.829	0.0569	0.0549
	1	0.7811	1.7071	1.3335	0.8536
3	0	0.0317	2.0633	0.0654	0.0619
	1	1.1667	1.9257	2.2468	0.9629
4	0	0.032	2.1271	0.068	0.0638
	1	1.6234	1.9853	3.223	0.9927
5	0	0.032	2.1403	0.0686	0.0642
	1	2.1115	1.9976	4.218	0.9988

Experimento 2

En este experimento se han introducido un tiempo de reflexión de 8 segundos y un total de 10 trabajos. Además, las razones de visita y los tiempos de servicio de cada dispositivo son 8 y 0,03s para el dispositivo 0 y 7 y 0,1s para el dispositivo 1, respectivamente.

Trabajos	R (s)	X (trabajos/s)
1	0.94	0.1119
2	1.0013	0.2222
3	1.0705	0.3307
4	1.1493	0.4372
5	1.2394	0.5412
6	1.343	0.6422
7	1.4626	0.7398
8	1.6013	0.8332
9	1.7628	0.9219
10	1.9511	1.0049

Trabajos	Dispositivo	Ri (s)	Xi (trabajos/s)	Ni (trabajos)	Ui (%)
1	0	0.03	0.8949	0.0268	0.0268
	1	0.1	0.783	0.0783	0.0783
2	0	0.0308	1.7775	0.0548	0.0533
	1	0.1078	1.5553	0.1677	0.1555
3	0	0.0316	2.6459	0.0837	0.0794
	1	0.1168	2.3152	0.2703	0.2315
4	0	0.0325	3.4975	0.1137	0.1049
	1	0.127	3.0603	0.3888	0.306
5	0	0.0334	4.3293	0.1446	0.1299
	1	0.1389	3.7881	0.5261	0.3788
6	0	0.0343	5.1376	0.1764	0.1541
	1	0.1526	4.4954	0.686	0.4495
7	0	0.0353	5.9181	0.2089	0.1775
	1	0.1686	5.1783	0.8731	0.5178
8	0	0.0363	6.6658	0.2417	0.2
	1	0.1873	5.8326	1.0925	0.5833
9	0	0.0373	7.375	0.2747	0.2212
	1	0.2092	6.4531	1.3503	0.6453
10	0	0.0382	8.0393	0.3074	0.2412
	1	0.235	7.0344	1.6533	0.7034

Experimento 3

En este experimento se han introducido un tiempo de reflexión de 8 segundos y un total de 10 trabajos. Además, las razones de visita y los tiempos de servicio de cada dispositivo son 8 y 0,03s para el dispositivo 0, 7 y 0,1s para el dispositivo 1 y 16 y 0.15s para el dispositivo 2, respectivamente.

Trabajos	R (s)	X (trabajos/s)
1	3.34	0.0882
2	3.8962	0.1681
3	4.6107	0.2379
4	5.5256	0.2957
5	6.6816	0.3406
6	8.1065	0.3725
7	9.8026	0.3932
8	11.7407	0.4053
9	13.8674	0.4116
10	16.1217	0.4146

Trabajos	Dispositivo	Ri (s)	Xi (trabajos/s)	Ni (trabajos)	Ui (%)
1	0	0.03	0.7055	0.0212	0.0212
	1	0.1	0.6173	0.0617	0.0617
	2	0.15	1.4109	0.2116	0.2116
2	0	0.0306	1.345	0.0412	0.0403
	1	0.1062	1.1768	0.1249	0.1177
	2	0.1817	2.6899	0.4889	0.4035
3	0	0.0312	1.9031	0.0594	0.0571
	1	0.1125	1.6653	0.1873	0.1665
	2	0.2233	3.8063	0.8501	0.5709
4	0	0.0318	2.3659	0.0752	0.071
	1	0.1187	2.0702	0.2458	0.207
	2	0.2775	4.7318	1.3131	0.7098
5	0	0.0323	2.7245	0.0879	0.0817
	1	0.1246	2.3839	0.297	0.2384
	2	0.347	5.449	1.8906	0.8174
6	0	0.0326	2.9802	0.0973	0.0894
	1	0.1297	2.6076	0.3382	0.2608
	2	0.4336	5.9603	2.5844	0.894
7	0	0.0329	3.1456	0.1035	0.0944
	1	0.1338	2.7524	0.3683	0.2752
	2	0.5377	6.2912	3.3825	0.9437

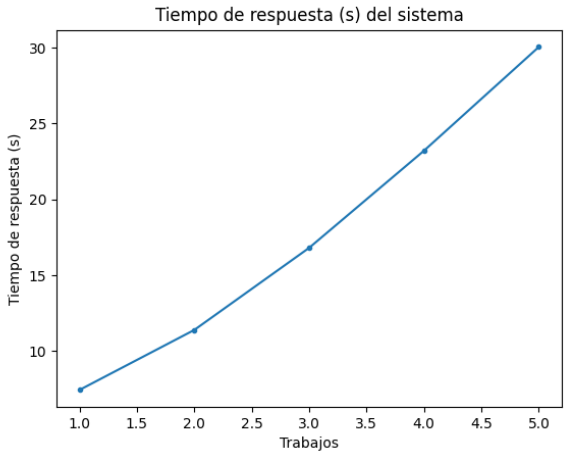
8	0	0.0331	3.242	0.1073	0.0973
	1	0.1368	2.8368	0.3882	0.2837
	2	0.6574	6.4841	4.2625	0.9726
9	0	0.0332	3.2926	0.1094	0.0988
	1	0.1388	2.881	0.3999	0.2881
	2	0.7894	6.5851	5.1981	0.9878
10	0	0.0333	3.3165	0.1104	0.0995
	1	0.14	2.902	0.4063	0.2902
	2	0.9297	6.633	6.1669	0.995

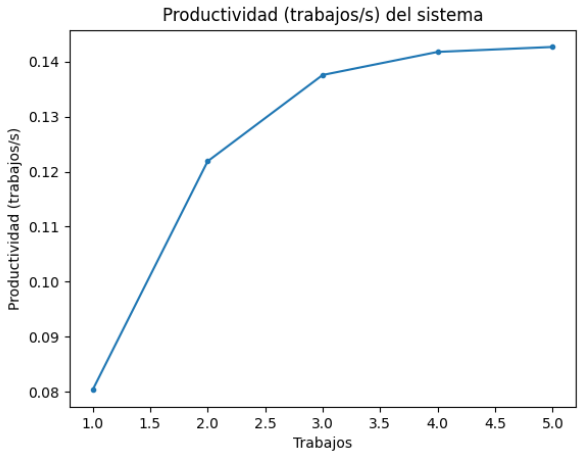
Nótese que en las tablas de los diferentes experimentos, el dispositivo 0, tiene como resultado valores bastante inferiores con respecto al dispositivo 0. Esto se debe a que el tiempo de servicio del primer dispositivo es mucho menor que del segundo, la cual cosa veremos reflejada en las gráficas a continuación.

Con todos estos experimentos, tendremos diferentes datos para poder observar el comportamiento del algoritmo. En el siguiente apartado se procederá a graficar y analizarlos.

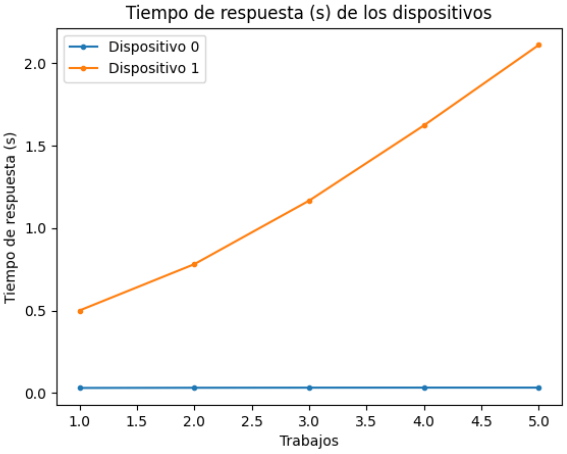
3. Añadir a la documentación la representación gráfica de las variables calculadas y una discusión sobre su comportamiento.

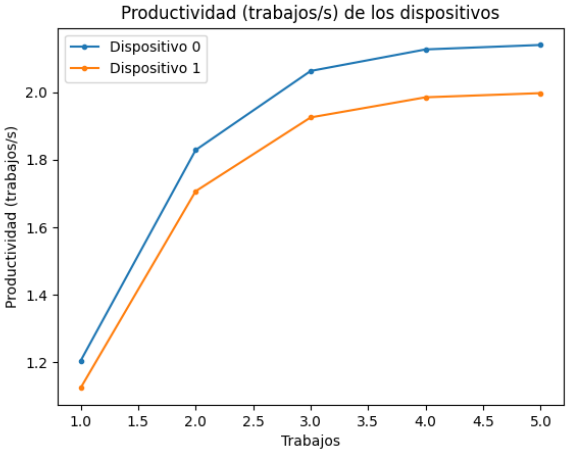
Experimento 1





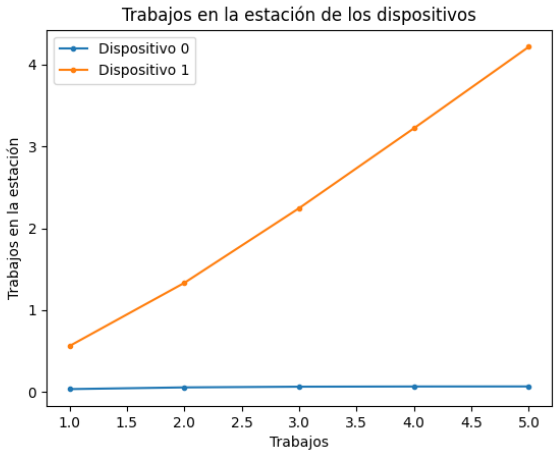
En primer lugar, podemos observar que el tiempo de respuesta muestra una tendencia lineal a largo plazo. Por tanto, podemos decir que a mayor número de trabajos, proporcionalmente el sistema tendrá un tiempo de respuesta mayor. Por otro lado, la productividad tiene una tendencia logarítmica, la cual nos muestra a partir de cuantos trabajos el sistema queda estancado. Si nos fijamos, a partir de 4 trabajos, el sistema ya se estanca en cuanto a productividad; pero realmente solo ha llegado a aproximadamente 0.14 trabajos/s. Eso nos indicia que algún dispositivo hace de cuello de botella.

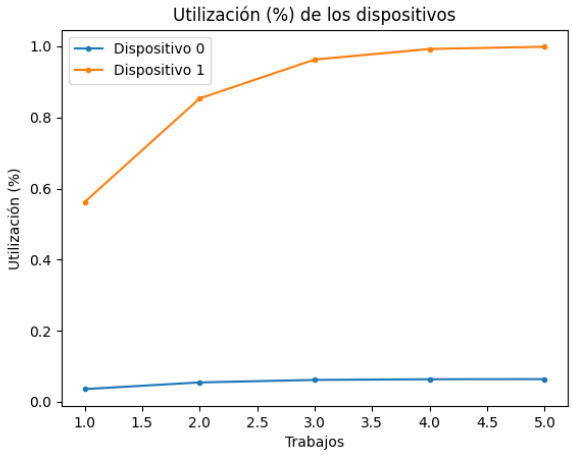




En este caso, como se ha comentado en el apartado anterior, observamos que los valores del dispositivo 0 en algunas gráficas, no se pueden apreciar del todo. De todos modos, es más interesante comparar los distintos valores entre dispositivos, con el objetivo de visualizar cuál está teniendo más o menos carga de trabajo. Es por eso que las tendencias del dispositivo 0 serán similares a las del dispositivo 1. Además, los resultados que nos pueda ofrecer un sistema al monitorizar los dispositivos, será similar a como están graficados ahora. Procedemos ahora si, a comentar su comportamiento.

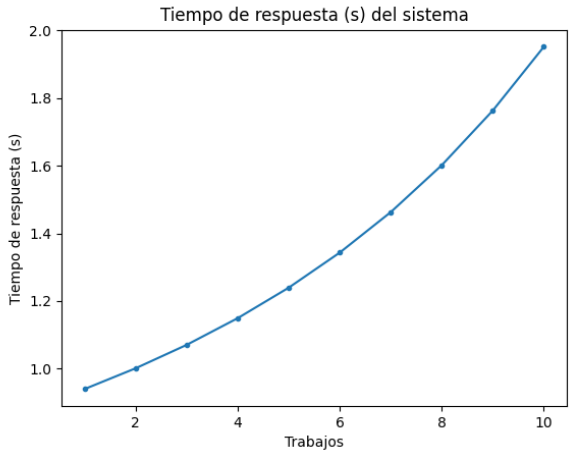
En primer lugar, observamos que el tiempo de respuesta del dispositivo 1 es mucho mayor (debido a los valores del tiempo de servicio de ambos) y parece mostrar una tendencia lineal relativamente suave. Esto nos indica que de primeras, con un número de trabajos bajos, la productividad es alta, pero a la larga se acaba estancando (a partir de 3) y a cuantos más trabajos, proporcionalmente tardara más tiempo. Estas suposiciones se afirman observando la gráfica de la productividad, que a partir de 3 trabajos, el sistema empieza a observar un cuello de botella. Comentar además, que es curioso como el dispositivo 0 es más productivo que el 1, la cual cosa a simple vista no nos lo parece. En consecuencia, es muy seguro que en las siguientes graficamos veamos como en las respectivas utilizaciones de los dispositivos, el 1 es el que nos perjudica el rendimiento general del sistema.

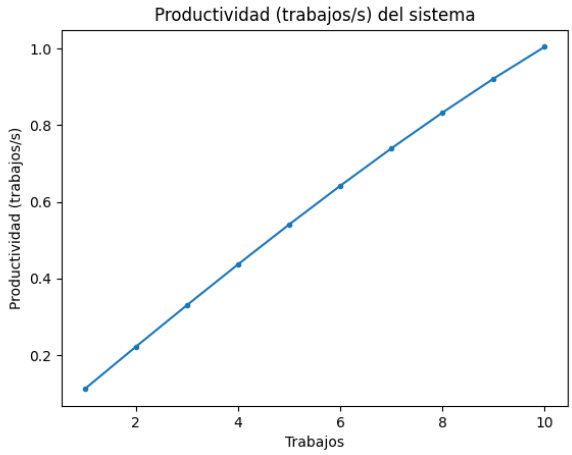




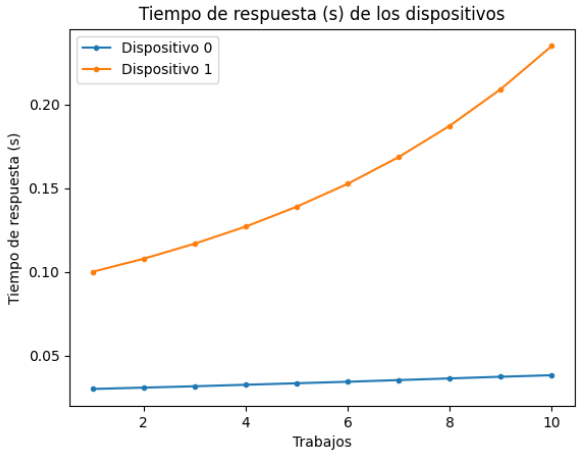
Aquí podemos ver como el número de trabajos que se van acumulando en la estación de cada dispositivo, aumenta linealmente. Esto se debe a que la llegada de trabajos también es lineal. Por otro lado, la utilización de los dispositivos es distinta. El dispositivo 1 llega a su punto de saturación a partir del trabajo 4, es decir, a su máxima utilización. En cambio, el dispositivo 0 también se estanca entre el trabajo 3 y 4, pero no pasa de un 10 % de utilización. Esto se debe a que el dispositivo 1 tiene una mayor carga de trabajo y, por tanto, una mayor demanda de servicio; es por eso que podríamos decir que se trata del cuello de botella dels sistema.

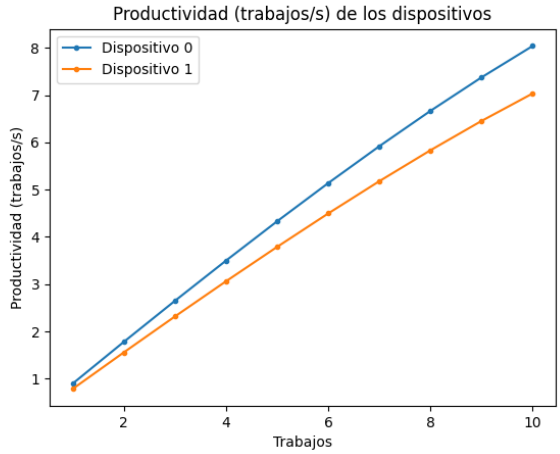
Experimento 2



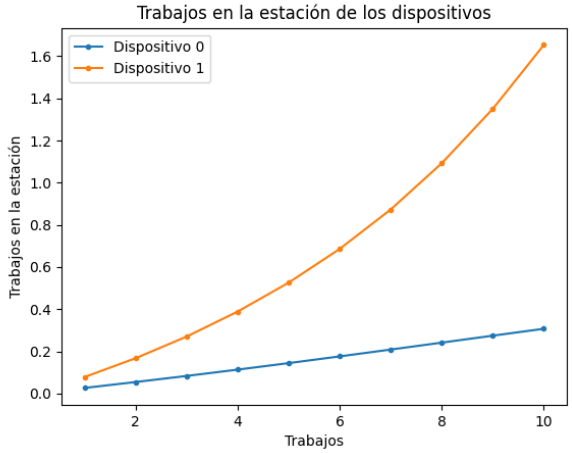


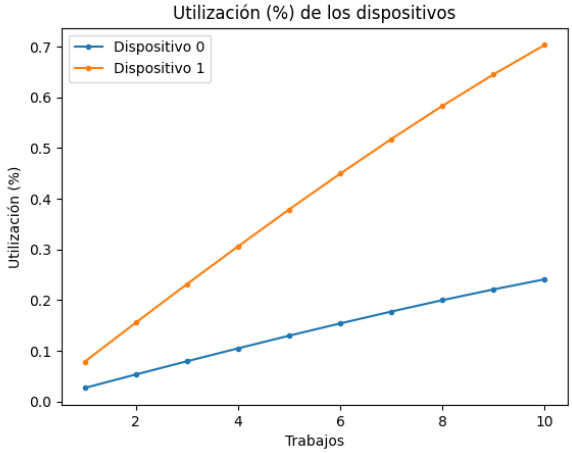
En este caso, el tiempo de respuesta del sistema tiene una tendencia exponencial. Esto significa que a mayor número de trabajos, el tiempo de respuesta es cada vez mayor (por tanto, peor que en el anterior experimento). Por otro lado, la productividad muestra una tendencia logarítmica, aunque no muy marcada. Podríamos deducir que a largo plazo se llegaría a estancar; pero con los datos recogidos no podemos deducir ese valor. En consecuencia, el sistema tiene aún tiene margen para soportar una mayor carga.





En este caso, los dispositivos se comportan muy diferente por lo que se refiere a su tiempo de respuesta. El dispositivo 0 tiene una tendencia lineal y, sin embargo, el dispositivo 1 la tiene exponencial. Esto significa que el dispositivo 0, a un mayor número de trabajos, tiene un menor tiempo de respuesta y en consecuencia es más rápido. Esto se ve reflejado en la gráfica de las productividades. Claramente, el dispositivo 0 realiza más trabajos por segundo que el dispositivo 1. Ambas no muestran una tendencia clara, pero por lo que nos dice la experiencia, si la calculáramos con un número mayor de trabajos, resultaría ser una tendencia logarítmica.

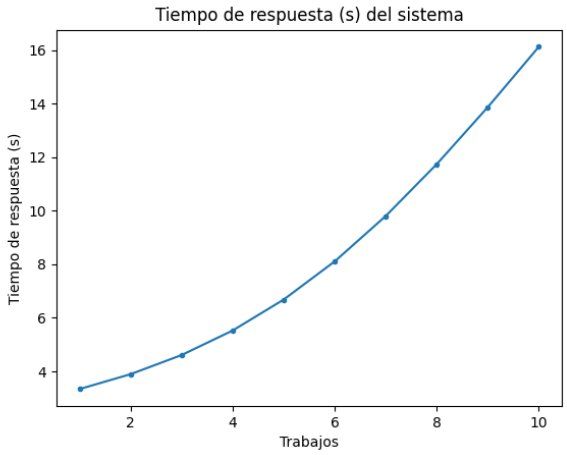


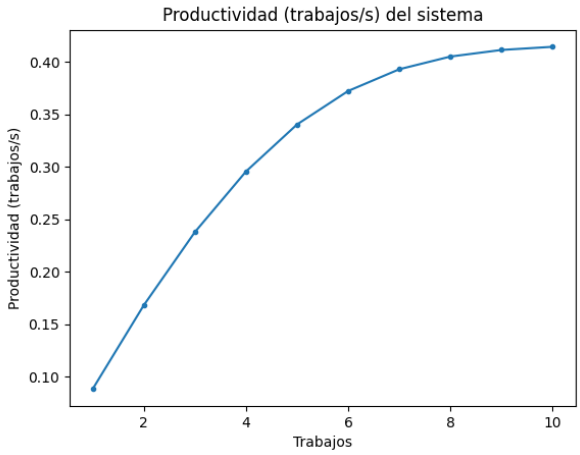


En este caso, nos encontramos que el dispositivo 1 también muestra peores resultados en estas gráficas. Por un lado, la tendencia de trabajos que se van acumulando en la estación es exponencial, frente a la lineal del dispositivo 0. Por otro lado, las dos utilizaciones muestran la misma tendencia, pero la del dispositivo 1 aumenta mucho más rápido en el eje de las ordenadas.

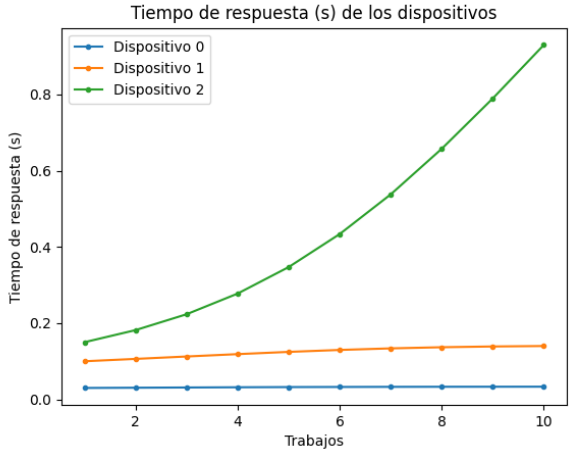
Entonces podemos concluir que en este experimento, ambos dispositivos tienen margen para recibir más carga de trabajo; ya que ninguno se encuentra saturado. Por consiguiente, el sistema tambien tiene margen para recibir más trabajo.

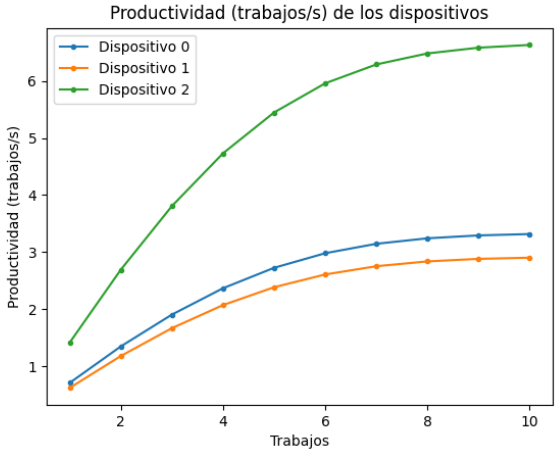
Experimento 3



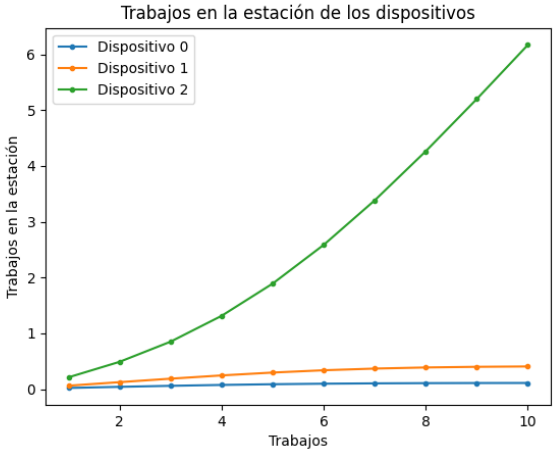


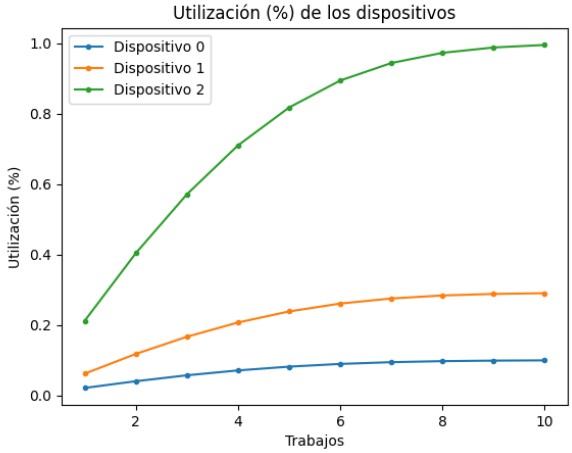
En este experimento, el tiempo de respuesta también es exponencial, y todo lo que ello conlleva (comentado en el experimento anterior). Además, la productividad del sistema muestra una tendencia logarítmica y a partir de los 10 trabajos aproximadamente, el sistema se estanca. Esto quiere decir que algunos de los dispositivos es nuestro cuello de botella.





Nótese que en estas gráficas hemos realizado el experimento con tres dispositivos, a diferencia de los otros que eran con dos. En este caso, por lo que hace al tiempo de respuesta, el dispositivo 0 y 1 tienen una tendencia similar. Sin embargo, el dispositivo 2 muestra una tendencia exponencial, por tanto, tiene un tiempo de respuesta mucho mayor que los otros. En consecuencia, con los mismos trabajos, empleará mucho más tiempo que los otros dispositivos. Como siempre, esto se ve reflejado en la productividad del sistema. Sin embargo, por lo que hace a la productividad, el dispositvo 2 muestra una diferencia notoria entre los otros dispostivos. Por mucho que el tiempo de respuesta del dispositvo 2 sea mucho mayor, su productividad sigue siendo mucho mejor que la de los otros dispositivos y esto se verá reflejado en la producitividad del sistema.





El dispositivo 2 muestra un mayor numero de trabajos en la estación en función del tiempo, frente a los otros dispisitivos. Concretamente, tiene una tendencia que primeramente parece exponencial pero se consigue fijar como una tendencia lineal (a un numero mayor de trabajos). Por otro lado, la utilización sigue la misma tendencia logaritmica en en todos los dispositivos, pero claramente la del dispositivo 2 llega practicmante al 100 % y por ende seria el cuello de botella del sistema.

Bloque 2 (Opcional)

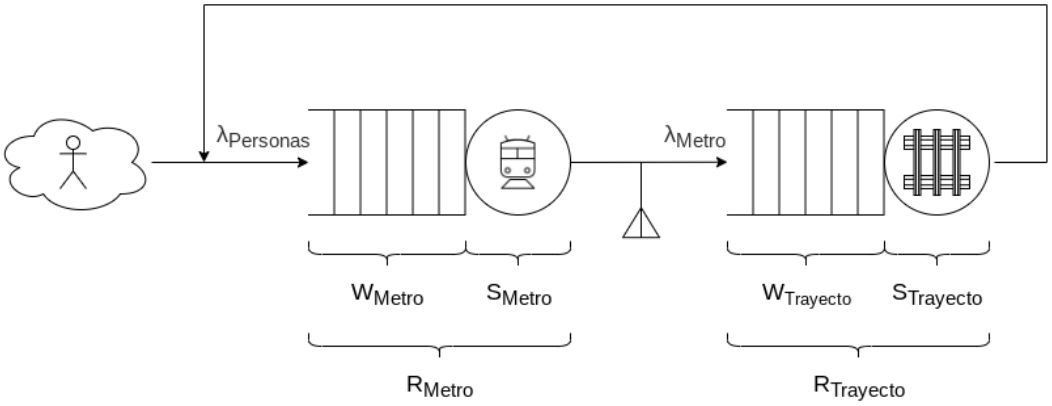
Dada la línea 11 del metro de Barcelona, la cual consta de 5 paradas, se pide modelar el sistema de transporte haciendo uso de la teoría de colas y responder a las siguientes preguntas:

La información necesaria para responder a las preguntas es:

- Frecuencia de llegada de pasajeros a cada estación: 20 pasajeros / minuto
- Frecuencia de llegada del metro en cada estación: 1 metro / 3 minutos
- Tiempo medio del recorrido entre estaciones: 1 minuto
- Probabilidad de un pasajero de bajarse: 40
- Tiempo medio de permanencia de un metro en la estación: 30 segundos

---

Para realizar esta parte de la práctica, se ha llevado a cabo un modelo de una posible solución del problema propuesto. El modelo en cuestión es el siguiente:



En este modelo, modelamos dos colas de forma recursiva (para todas las estaciones del metro). La primera cola sería una cola abierta, que representaría la llegada de personas a la estación de metro; y su servicio sería entrar dentro del vagón del metro. Por otro lado, tenemos la cola que modela los trenes del metro, concretamente el trayecto que estos siguen. Como la red de metros es una red cerrada, la respectiva cola también lo será y su servicio sería el tiempo que tarda en hacer un trayecto de una estación a otra.

Por tanto, con los datos que se no facilitan, podemos deducir las siguientes equivalencias:

- Frecuencia de llegadas de pasajeros a cada estación = $\lambda_{Personas}$
- Frecuencia de llegada del metro en cada estación = λ_{Metro}
- Tiempo del recorrido entre estaciones = $S_{Trayecto}$
- Tiempo medio de permamencia de un metro en la estación = $S_{Metro} = W_{Trayecto}$

Ahora entonces, con el modelo hecho y teniendo claro que es cada cosa, podemos proceder a resolver las siguientes cuestiones.

1. ¿Cuánto tiempo tarda un metro en recorrer la línea?

Siguiendo el modelo anterior, el tiempo que tarda en recorrer el metro la línea sería equivalente al tiempo de respuesta de la cola cerrada que representa el trayecto. El tiempo de servicio nos lo dan, es 1 minuto, y el tiempo de espera será el tiempo de servicio de la cola anterior, es decir, 30 segundos. Por tanto, el tiempo de respuesta se puede resolver de la siguiente forma:

$$R_{Trayecto} = S_{Metro} + S_{Trayecto} = W_{Trayecto} + S_{Trayecto} = 30s + 1min = 1,5min$$

2. ¿Cuál es la probabilidad de llegar a la estación y encontrarse un metro? ¿Y qué no haya?

En este caso entendemos que la probabilidad de que un cliente encuentre un metro en la estación, sería equivalente a cuando el metro está siendo utilizado.

$$P_{Metro} = U_{Metro} = \lambda_{Metro} * S_{Metro} = \frac{1}{180} trabajos/s * 30s = \frac{1}{6} = 0,1667$$

Por tanto, la probabilidad de llegar y encontrarse un metro es de un 16,67 %. En consecuencia, la probabilidad de no encontrarselo seria $1 - 0,1667 = 0,8333$, es decir, un 83,33 %.

3. ¿Cuál es la frecuencia de llegadas máxima que soporta el sistema?

La frecuencia máxima que soportara el sistema será la frecuencia máxima que soporte la segunda cola; ya que realmente el cuello de botella del sistema es el trayecto que realizan los metros. En este caso, se rige por la siguiente fórmula:

$$\lambda_{max} = \frac{1}{D_b}$$

Pero para caluclar la D_b necesitamos saber las razones de visita del dispositivo. En este caso es 1, ya que las vías del tren solo pueden llevar un metro. Entonces, la ecuación se resolverá de la siguiente forma:

$$\lambda_{max} = \frac{1}{D_b} = \frac{1}{V * S} = \frac{1}{1 * 30} = 0,0333$$

Por tanto, la mayor frecuencia de metros/segundo que soportara el sistema será de 0,0333, es decir, aproximadamente 120 metros cada hora.

Scripts

A continuación el script implementado:

```
import csv
import matplotlib.pyplot as plt

# Leemos el fichero con los datos
with open("data.txt", "r") as f:
    csv_reader = list(csv.reader(f, delimiter=" "))

# Eliminamos la cabecera de las columnas
csv_reader.pop(0)

# Declaramos la razón de visita y el tiempo de servicio
V = []
S = []

# Almacenamos los datos del fichero en los arrays correspondientes
for i in range(len(csv_reader)):
    V.append(float(csv_reader[i][0]))
    S.append(float(csv_reader[i][1]))

dispositivos = len(V)

# Array bidimensional que contendra los resultados obtenidos
resultados_sistema = []
resultados_dispositivos = []

# Almacenamos el input del usuario
N = int(input("Introduce el número de trabajos: "))
Z = int(input("Introduce el tiempo de reflexión: "))


def __main__():
    """ Algoritmo para el análisis del valor medio para redes de colas
    ↪  cerradas"""

    # Para todos los trabajos
    for n in range(1, N + 1):
        print(f"----- Job {n} -----")


```

```
        # Calculamos el tiempo de respuesta y la productividad del sistema
        R = formatear(calcularR(n))
        X = formatear(calcularX(n))

        resultados_sistema.append([n, R, X])

        # Para todos los dispositivos
        for i in range(dispositivos):
            Ri = formatear(calcularRi(n, i))
            Xi = formatear(calcularXi(n, i))
            Ni = formatear(calcularNi(n, i))
            Ui = formatear(calcularUi(n, i))

            resultados_dispositivos.append([i, Ri, Xi, Ni, Ui])

    # Almacenamos los resultados del sistema en un fichero
    with open("resultados_sistema.txt", "w") as f:
        write = csv.writer(f)
        write.writerows(resultados_sistema)

    # Almacenamos los resultados de los dispositivos en un fichero
    with open("resultados_dispositivos.txt", "w") as f:
        write = csv.writer(f)
        write.writerows(resultados_dispositivos)

    # Almacenamos los valores del eje x para todas las graficas
    j = 0
    eje_x = [fila[j] for fila in resultados_sistema]

    # Almacenamos las cabeceras de los valores del eje y
    cabeceras = [
        "Tiempo de respuesta (s)",
        "Productividad (trabajos/s)"]

    cabeceras_i = [
        "Tiempo de respuesta (s)",
        "Productividad (trabajos/s)",
        "Trabajos en la estación",
        "Utilización (%)"]

    # Graficamos los resultados del sistema
    for i in range(len(cabeceras)):
        plt.figure()

```

```
        # Obtenemos los valores del eje y
        eje_y = [fila[i + 1] for fila in resultados_sistema]

        # Graficamos los resultados
        plt.plot(eje_x, eje_y, marker = 'o', markersize = 3)

        # Añadimos las etiquetas
        plt.xlabel("Trabajos")
        plt.ylabel(f"{cabeceras[i]}")
        plt.title(f"{cabeceras[i]} del sistema")
        plt.savefig(f"grafica_sistema_{i}.png")

    # Graficamos los resultados de los dispositivos
    for i in range(len(cabeceras_i)):
        plt.figure()

        # Obtenemos los valores del eje y de cada dispositivo
        for j in range(dispositivos):
            eje_y = [fila[i + 1] for fila in resultados_dispositivos if fila[0]
            ↪  == j]

            # Graficamos los resultados
            plt.plot(eje_x, eje_y, label=f"Dispositivo {j}", marker = 'o',
            ↪  markersize = 3)

        # Añadimos las etiquetas
        plt.xlabel("Trabajos")
        plt.ylabel(f"{cabeceras_i[i]}")
        plt.title(f"{cabeceras_i[i]} de los dispositivos")
        plt.legend(loc='upper left')
        plt.savefig(f"grafica_{i}.png")

# Función para formatear los resultados
def formatear(x):
    return float(('%.4f' % x).rstrip('0').rstrip('.'))

# Función para calcular el tiempo de respuesta
def calcularR(n):
    return sum(V[i] * calcularRi(n, i) for i in range(dispositivos))

# Función para calcular la productividad del sistema
def calcularX(n):
    return n / (Z + calcularR(n))

```

```
126 # Función para calcular la productividad de un dispositivo
127 def calcularXi(n, i):
128     return calcularX(n) * V[i]
129 
130 # Función para calcular el número de trabajos de un dispositivo
131 def calcularNi(n, i):
132     return (calcularX(n) * V[i] * calcularRi(n, i) if n != 0 else 0)
133 
134 # Función para calcular el tiempo de respuesta de un dispositivo
135 def calcularRi(n, i):
136     return (calcularNi(n - 1, i) + 1) * S[i]
137 
138 # Función para calcular la utilización de un dispositivo
139 def calcularUi(n, i):
140     return calcularX(n) * V[i] * S[i]
141 
142 
143 if __name__ == '__main__':
144     __main__()
```

Práctica 6

El objetivo de esta práctica es la comprensión del concepto de caracterización de la carga. Para ello, se hará uso de la herramienta Weka.

De la monitorización de un sistema de almacenamiento, se proporciona un fichero de datos llamado data.txt. En el fichero se almacenan tres columnas con la siguiente información:

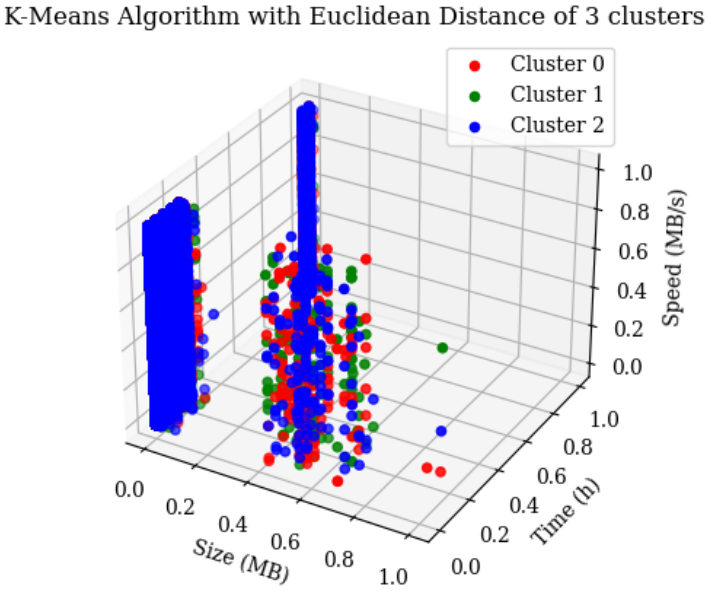
- El tamaño del fichero accedido (en MB). Los valores que correspondan con "-1" quieren decir que el acceso al fichero ha fallado.
- La hora a la que se hizo el acceso. El valor 22 representan las 22h, el valor 01 representan las 1h (a.m.), etc.
- El ancho de banda consumido (en MS/s). Los valores de esta columna están entre 453 y 1355, por lo tanto, los valores de esta columna deberán ser tratados. Es decir, el valor crudo de "1258.84,", corresponde con "1258,84".

Con los datos proporcionados se pide caracterizar la carga haciendo uso del algoritmo de K-Means y responder a las siguientes preguntas:

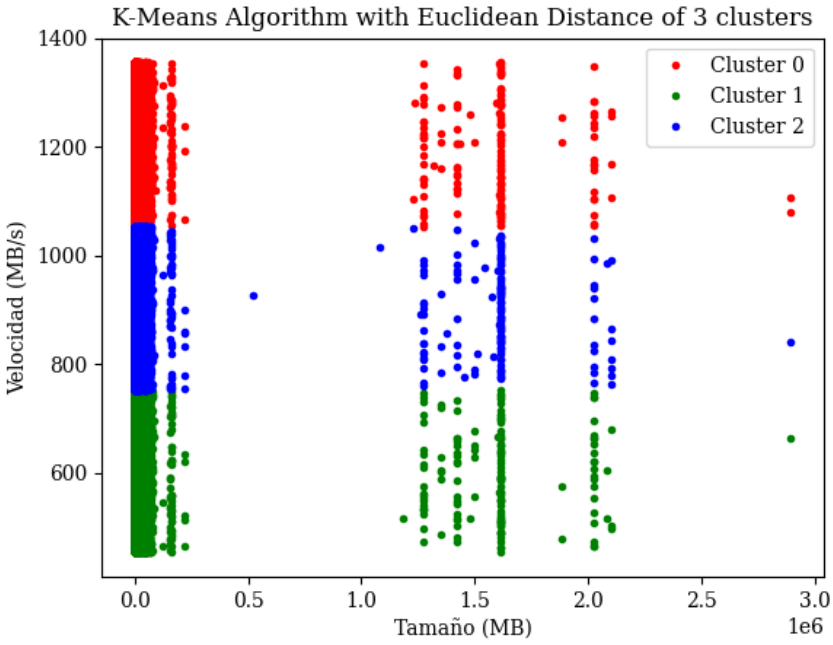
Para realizar el correcto filtrado de los datos del fichero, se ha realizado el script para que el software Weka pueda interpretar de forma correcta los datos; queda adjunto al final de la práctica.

1. Aplicando el algoritmo con 100 iteraciones y agrupando los datos en 3 clases, ¿qué resultados se obtienen? Muéstralo gráficamente.

En primer lugar, se llevó a cabo la implementación de un script (opcional) que, teniendo como *input* los datos ofrecidos por el Weka, genera una gráfica en tres dimensiones (ya que disponíamos de 3 variables) para obtener una correcta visualización de las proyecciones de los datos.



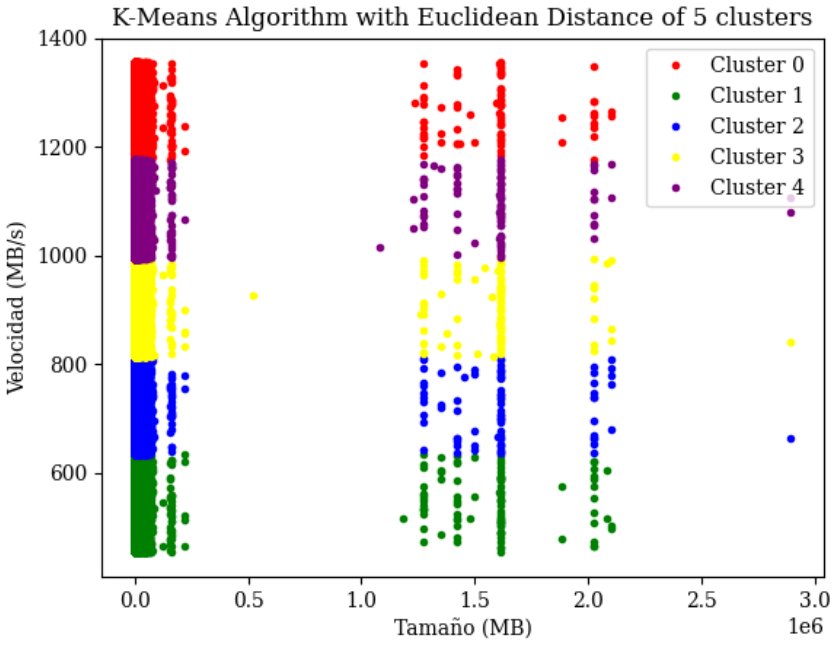
Entonces, al visualizar esta gráfica, rápidamente nos damos cuenta de que la información que nos da es un tanto extraña. Es posible esperar otro tipo de agrupamientos, pero analizándolo detalladamente, nos percatamos de que esto se debe a que el eje del *Time (h)* no nos está dando ninguna información relevante. Al no tener datos de todo el día, dicha variable no nos es del todo útil y, por tanto, se puede despreciar. Es por ello que ahora si podemos graficar los resultados conforme a la gráfica siguiente; la cual representa, en dos dimensiones, el tamaño frente a la velocidad de transferencia.



Observamos ahora que el Weka nos agrupa los datos en tres grupos (como se le había pedido) proporcionalmente iguales. Sin embargo, no tenemos una nube de puntos, la gráfica sigue un comportamiento conocido como o *heavy-tailed distribution*. Este comportamiento representa una gran cantidad de valores próximos al punto cero de la gráfica y conforme avanza la gráfica se va reduciendo de forma drástica el número de valores. Esto nos puede hacer pensar que el sistema de almacenamiento mueve muchos más ficheros de un tamaño pequeño que de tamaños más grandes. Es obvio ya que los sistemas operativos suelen contener una multitud de ficheros de configuración o de registros, que suelen pesar bastante poco. Y claramente son ficheros que se van actualizando constantemente.

2. Con el mismo número de iteraciones y agrupando los datos en 5 clases, ¿qué resultados se obtienen? ¿Cómo difieren de los anteriormente obtenidos?

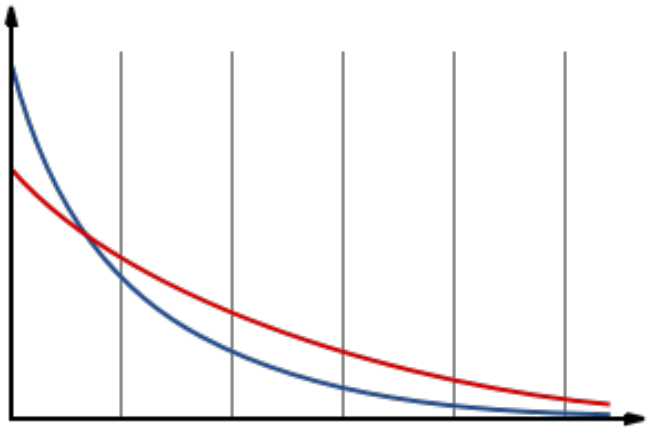
Aquí también se realizó la representación en tres dimensiones. Pero como se puede presuponer (viendo los resultados anteriores) tampoco nos va a dar la información de una forma útil para analizarla. Por tanto, también se han realizado los mismos pasos que en el apartado anterior y se ha representado en dos dimensiones directamente.



Vemos que sigue el mismo patrón que en apartado anterior. Es decir, observamos el mayor número de transferencias de ficheros, son los que ocupan menos y a medida que el tamaño aumenta, se realizan menos transferencias. La única diferencia palpable es que el Weka ha realizado cinco clústeres en vez de tres.

3. ¿Hay alguna característica especial en la carga proporcionada? Explícala con detalle.

Sí, encontramos unas cuantas. En primer lugar, es un poco extraño que el agrupamiento de los datos por clúster no sea una nube de puntos, como se había visto en las clases teóricas. De hecho, dicho agrupamiento ha sido bastante igualado entre clústeres. Esto nos indica que los datos introducidos en el Weka tal vez no eran los correctos, para lo que finalmente queríamos observar y exige una comprensión extra por parte del analista para darse cuenta de qué está pasando. En segundo lugar, observamos que los clústeres se dividen en tres columnas (aunque podrían ser dos, dependiendo del zoom que hagamos o lo concretos que queramos ser), la cual cosa sorprende. Esto está ligado con la tercera característica especial, que es el porqué se generan estas columnas en los gráficos. Como se ha comentado un poco anteriormente, parece ser que los datos siguen una distribución de cola pesada (o *heavy-tailed distribution*). Este tipo de distribuciones se caracteriza por concentrar la mayor parte de los datos en los valores más cercanos al cero y van disminuyendo a medida que aumenta el eje de las abscisas. A continuación un ejemplo de dicha distribución.



Heavy-Tailed Distribution

En este ejemplo, la curva que representaría más la tendencia que siguen nuestros datos sería la de color azul. Si nos imagináramos que los ejes tuvieran los mismos nombres que en las gráficas (de dos dimensiones) anteriores, entonces veríamos claramente que se concentran la mayoría de ficheros al principio del gráfico.

Finalmente, comentar que para tener una mejor representación (y en consecuencia un mejor análisis) de los clústeres, se debería hacer un trabajo previo de agrupación cualitativa en función del tamaño de los ficheros. Esto nos permitiría realizar un clúster de cada agrupación de datos (en este caso posiblemente serian 3) y seguramente el Weka nos mostraría unos clústeres mucho más interesantes que los que nos ha mostrado hasta ahora.

Scripts

A continuación el script de filtrado de los datos que nos genera un archivo .arff de Weka:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("-i", "--input", help="Input file",
                    default="data.in", type=str)
parser.add_argument('-c', "--csv",   help="csv format?",
                    action=argparse.BooleanOptionalAction)
args = parser.parse_args()

def main():
    with open(args.input) as file:
        data = file.read().splitlines()

    # Remove header
    data.pop(0)
    with open(f"data.{'csv' if args.csv else 'arff'}", "x") as output:
        if args.csv:
            output.write("size,hour,MB/s\n")
        else:
            output.write("@relation data-server\n\n")
            output.write("@attribute SIZE numeric\n")
            output.write("@attribute HOUR numeric\n")
            output.write("@attribute MBS numeric \n")
            output.write("\n@data\n\n")

        for line in data:
            line = line.split(",")

            # Remove first -1
            if line[0] == "-1":
                continue

            # Remove last part
            try:
                line.pop(2)
            except:
                pass

            # Split values
            try:
```

```
                temp = line[1].split("\t")
            except:
                print(line)

            line[1] = temp[0]

            # Refractor numbers
            line.append(fractor_number(temp[1]))

            output.write(f"{line[0]},{line[1]},{line[2]}\n")

def fractor_number(number):
    temp = number.split(".")

    if len(temp) == 3:
        return f"{temp[0]}{temp[1]}.{temp[2]}"
    else:
        return number

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

A continuación el script que nos permite representar tantos clusters como genere el Weka, en tres dimensiones:

```
import argparse
import matplotlib.pyplot as plt
import numpy as np

colors = ("r", "g", "b")

parser = argparse.ArgumentParser()
parser.add_argument("-c", "--cluster", help="Select cluster number",
                    default=3, type=int)
args = parser.parse_args()


def main():
    # Get number of clusters from keyboard input
    num_clusters = args.cluster

    plt.figure()
    ax = plt.axes(projection='3d')

    # Get cluster data and make a scatter plot
    for i in range(num_clusters):
        x_norm, y_norm, z_norm = get_cluster_norm_values(i)
        ax.scatter3D(x_norm, y_norm, z_norm,
                     label=f"Cluster {i}", c=colors[i], marker="o")

    # Plot data
    ax.set_title(
        f"Representation of {args.cluster} clusters | K-Means Clustering with
        ↪  Euclidean Distance")
    ax.set_xlabel('Size (MB)')
    ax.set_ylabel('Time (h)')
    ax.set_zlabel('Speed (MB/s)')
    ax.legend(bbox_to_anchor=(1.05, 1), ncol=num_clusters)
    plt.savefig("cube.png")


def get_cluster_norm_values(value):
    with open("cluster.arff", "r") as file:
        data = file.read().splitlines()

    # Remove header
```

```
    data = data[9:]

    # Create an array with the cluster data
    cluster_value = []
    for line in data:
        line = line.split(',')
        if line[4] == f"cluster{value}":
            cluster_value.append([line[1], line[2], line[3]])

    # Create diferent arrays for each axis
    cluster_x = [float(x[0]) for x in cluster_value]
    cluster_y = [float(x[1]) for x in cluster_value]
    cluster_z = [float(x[2]) for x in cluster_value]

    # Get min and max values
    x_mn = [np.min(cluster_x), np.max(cluster_x)]
    y_mn = [np.min(cluster_y), np.max(cluster_y)]
    z_mn = [np.min(cluster_z), np.max(cluster_z)]

    # Normalize data
    x_norm = [normalize(data, x_mn[0], x_mn[1]) for data in cluster_x]
    y_norm = [normalize(data, y_mn[0], y_mn[1]) for data in cluster_y]
    z_norm = [normalize(data, z_mn[0], z_mn[1]) for data in cluster_z]

    # Return a tuple
    return x_norm, y_norm, z_norm


def normalize(value, min, max):
    return (value - min) / (max - min)


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

A continuación el script que nos permite representar tantos clusters como genere el Weka, en dos dimensiones:

```
import argparse
import matplotlib.pyplot as plt

colors = ("red", "green", "blue", "yellow", "purple")

parser = argparse.ArgumentParser()
parser.add_argument("-c", "--cluster", help="Select cluster number",
                    default=0, type=int)
args = parser.parse_args()

# Get number of clusters from keyboard input
num_clusters = args.cluster


def main():
    plt.figure()
    plt.rcParams.update({"font.family": "serif"})

    # Get cluster data and make a scatter plot
    for i in range(num_clusters):
        x_norm, y_norm = get_cluster_norm_values(i)
        plt.plot(x_norm, y_norm, 'o', color=colors[i], markersize=3,
        ↪  label=f"Cluster {i}")

    # Plot data
    plt.xlabel("Tamaño (MB)")
    plt.ylabel("Velocidad (MB/s)")
    plt.title(
        f"K-Means Algorithm with Euclidean Distance of {num_clusters}
        ↪  clusters")
    plt.legend(loc="upper right")
    plt.savefig(f"cluster_{num_clusters}.png")


def get_cluster_norm_values(value):
    with open(f"cluster{num_clusters}.arff", "r") as file:
        data = file.read().splitlines()

    # Remove header
    data = data[8:]

```

```
    # Create an array with the cluster data
    cluster_value = []
    for line in data:
        line = line.split(',')
        if line[4] == f"cluster{value}":
            cluster_value.append([line[1], line[3]])

    # Create diferent arrays for each axis
    cluster_x = [float(x[0]) for x in cluster_value]
    cluster_y = [float(x[1]) for x in cluster_value]

    # Return a tuple
    return cluster_x, cluster_y


def normalize(value, min, max):
    return (value - min) / (max - min)


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

Práctica 7

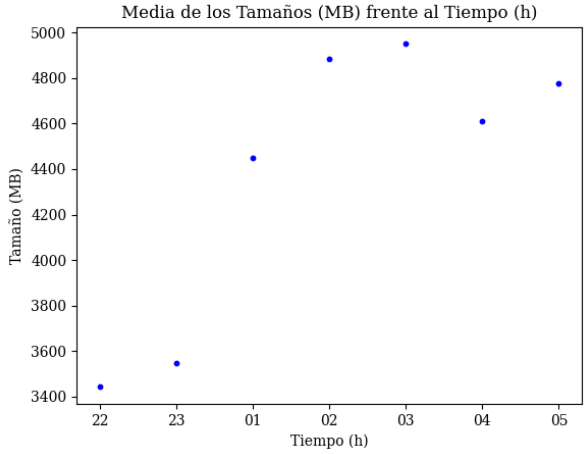
Una empresa de almacenamiento en la nube monitoriza la actividad de sus usuarios, es decir, se guarda la hora de acceso del cliente, el tamaño del fichero al que se ha accedido y la cantidad de información transmitida por unidad de tiempo (hacer uso de los datos de la práctica 6).

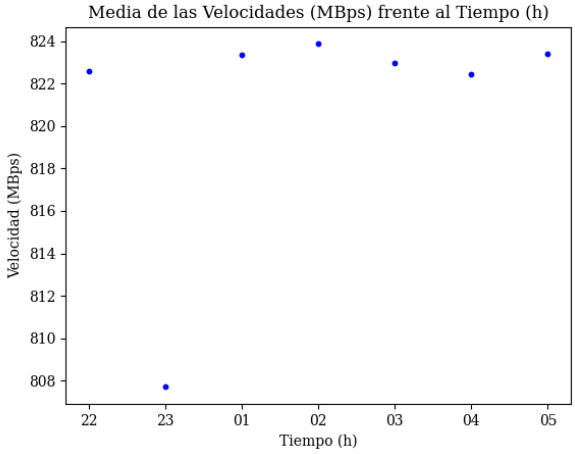
El director del departamento de informática de la empresa solicita calcular la cantidad de información transmitida por la red y el tamaño del fichero accedido para las 6 a.m. (recordar que la última hora monitorizada son las 5 a.m.).

- 1. ¿Qué patrón siguen los datos monitorizados? Proporciona una representación gráfica.
- 2. Calcula los valores solicitados para las 6 a.m. haciendo uso de la regresión lineal, medias móviles (usar los 4 últimos valores) y suavizado exponencial (peso fijo del 60 %).
- 3. ¿Qué técnica de predicción funciona mejor? ¿Por qué? ¿Cuál es la más adecuada para los datos con los que contamos?

Comentar que para la realización de esta práctica se ha implementado un script que realiza todos los cálculos necesarios. Este se encuentra adjunto al final del documento.

1. ¿Qué patrón siguen los datos monitorizados? Proporciona una representación gráfica.



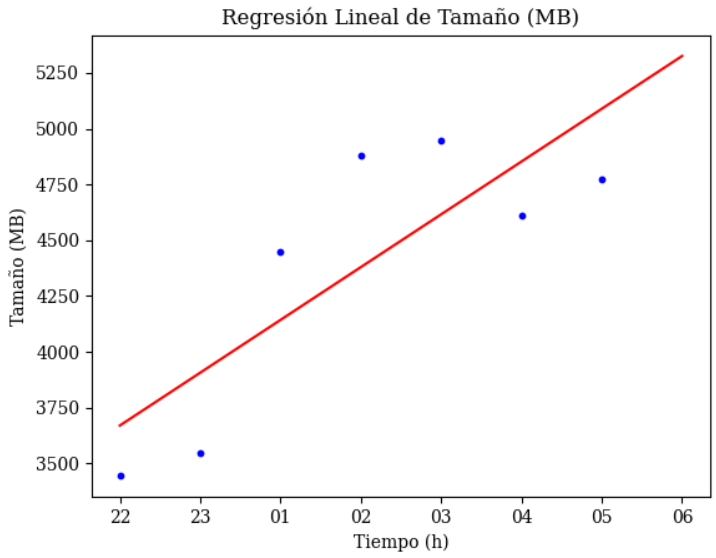


Por lo que hace al tamaño de los ficheros, podemos observar dos grupos de datos. Por un lado, los referentes al rango de horas entre las 22 y las 23, donde el sistema trabaja con ficheros de un tamaño relativamente pequeño. Sin embargo, el segundo grupo, referente al rango de horas entre la 1 y las 5 de la mañana, el sistema trabaja con ficheros de un tamaño mucho mayor. Esto suele ser habitual en los sistemas (tipo servidores) que, durante las horas de la noche, realizan los trabajos más pesados. Esto se hace así porque si se hicieran de día (mientras el usuario también hace uso del sistema) los usuarios podrían ser los perjudicados.

Por otro lado, tenemos la velocidad a la cual se transfieren dichos ficheros. Nótese que prácticamente durante toda la monitorización del sistema, este transfiere los archivos lo más rápido que puede, ya que prácticamente es el mismo valor en todo el eje x. Observamos, pero, una bajada importante a las 23h. Esto podría ser por el cambio de pasar a trabajar con ficheros de tamaño considerablemente mayor; ya que si nos fijamos en la gráfica anterior, dicho cambio se efectúa sobre la misma hora.

2. Calcula los valores solicitados para las 6 a.m. haciendo uso de la regresión lineal, medias móviles (usar los 4 últimos valores) y suavizado exponencial (peso fijo del 60 %).

Cálculos para el Tamaño



Para construir la recta que representara la regresion lineal se rige por la ecuación de la recta $y = a + bx$, donde

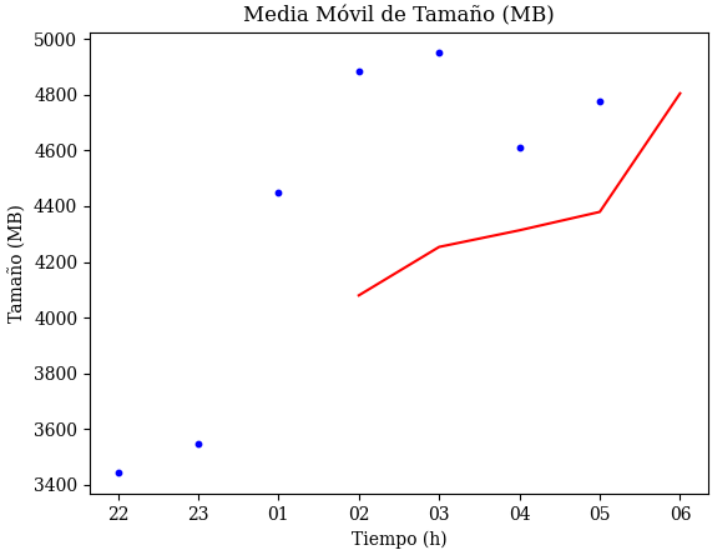
$$b = \frac{\sum_{i=1}^{n} x_i \times y_i - n \times \bar{x} \times \bar{y}}{\sum_{i=1}^{n} x_i^2 - n \times \bar{x}^2}$$

$$a = \bar{y} - b \times \bar{x}$$

Entonces, nos quedarian los siguientes resultados:
$y = 3669{,}2544 + 236{,}6961x$
$\bar{x} = 4$
$\bar{y} = 4379{,}3427$

Y por tanto, la prediccion de la hora 6 seria:

$$y = 3669{,}2544 + 236{,}6961 * (8) = 5562{,}8232MB$$

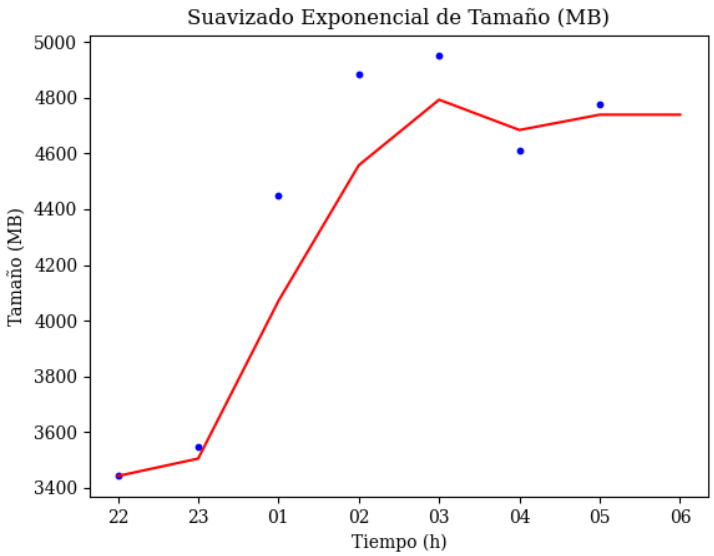


Dicho cálculo de las medias móviles viene dada por la siguiente ecuación:

$$f_{t+1} = \frac{y_t + y_{t-1} + ... + y_{t-n+1}}{n}$$

Donde f_{t+1} es la predicción, y_{tn} son los valores del eje y que queremos utilizar para realizar la prediccion y n que es el número de y_t que cojamos. Por tanto, la operación para calcular la sexta hora (con los últimos cuatro valores) es el siguiente:

$$f_6 = \frac{4881{,}8906 + 4949{,}225 + 4611{,}4016 + 4775{,}2293}{4} = 4804{,}4366MB$$



Dicho cálculo de las medias móviles viene dada por la siguiente ecuación:

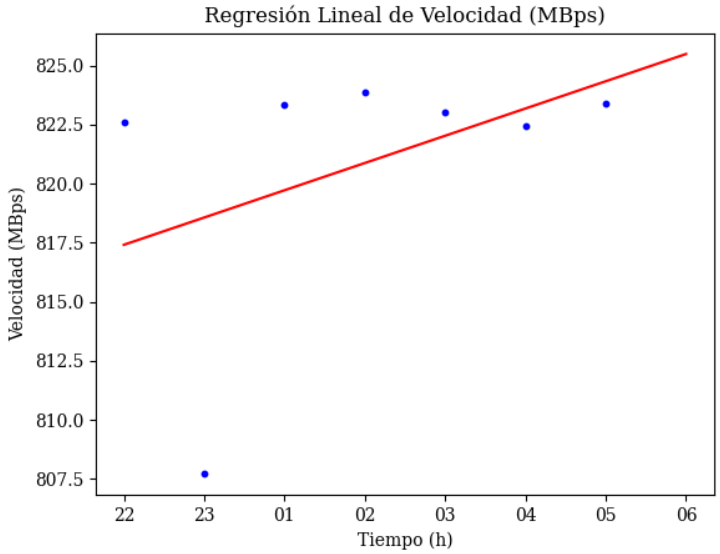
$$f_{t+1} = f_t + \alpha(y_{t+1} - f_t)$$

El cual se tendra que aplicar para cada valor de y. En este caso aplicaremos un $\alpha = 0,6$:

Hora	Media del tamaño (MB)	Predicción
1	3443.8934	3443.8934
2	3546.7858	3505.2288
3	4447.9735	4070.8756
4	4881.8906	4557.4846
5	4949.225	4792.5288
6	4611.4016	4683.8524
7	4775.2293	4738.6785

Y, por tanto, el valor que le corresponde a la hora 6 es, 4738.6785 MB.

Cálculos para la Velocidad



Para construir la recta que representara la regresion lineal se rige por la ecuación de la recta $y = a + bx$, donde

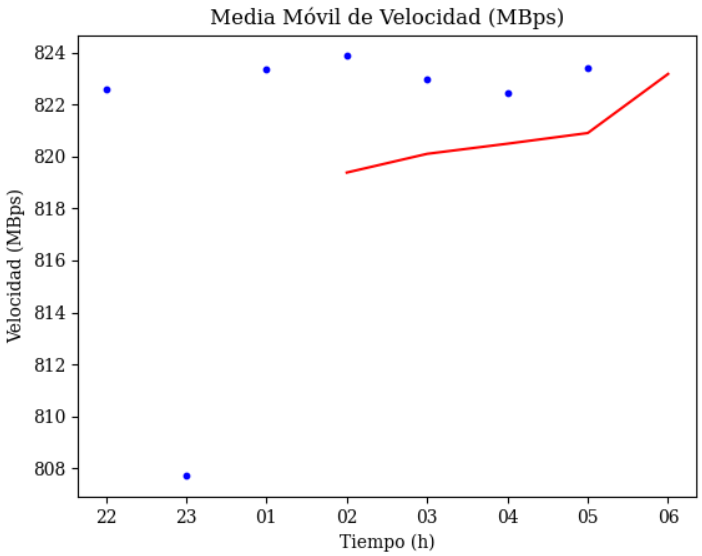
$$b = \frac{\sum_{i=1}^{n} x_i \times y_i - n \times \bar{x} \times \bar{y}}{\sum_{i=1}^{n} x_i^2 - n \times \bar{x}^2}$$

$$a = \bar{y} - b \times \bar{x}$$

Entonces, nos quedarian los siguientes resultados:
$y = 817{,}4075 + 1{,}1535x$
$\bar{x} = 4$
$\bar{y} = 820{,}8680$

Y por tanto, la prediccion de la hora 6 seria:

$$y = 817{,}4075 + 1{,}1535 * (8) = 826{,}6355\text{MBps}$$

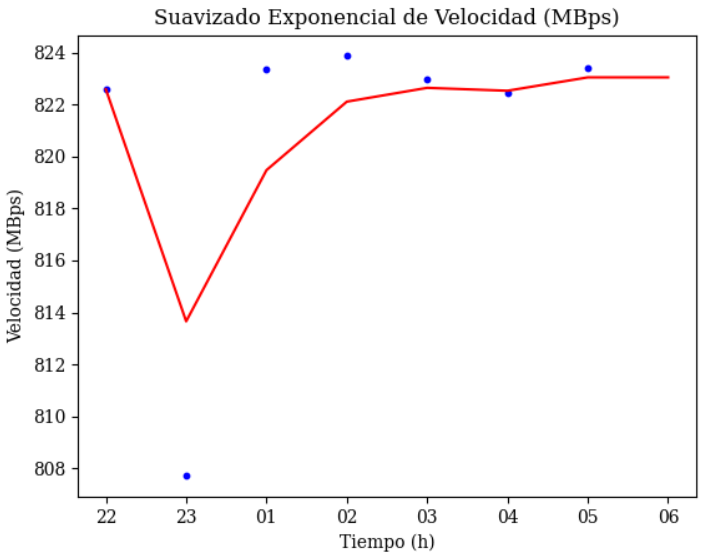


Dicho cálculo de las medias móviles viene dada por la siguiente ecuación:

$$f_{t+1} = \frac{y_t + y_{t-1} + ... + y_{t-n+1}}{n}$$

Donde f_{t+1} es la predicción, y_{tn} son los valores del eje y que queremos utilizar para realizar la prediccion y n que es el número de y_t que cojamos. Por tanto, la operación para calcular la sexta hora (con los últimos cuatro valores) es el siguiente:

$$f_6 = \frac{823{,}8627 + 822{,}9925 + 822{,}4515 + 823{,}3838}{4} = 823{,}1726\text{MBps}$$



Dicho cálculo de las medias móviles viene dada por la siguiente ecuación:

$$f_{t+1} = f_t + \alpha(y_{t+1} - f_t)$$

El cual se tendra que aplicar para cada valor de y. En este caso aplicaremos un $\alpha = 0,6$:

Hora	Media del tamaño (MB)	Predicción
1	822.5798	822.5798
2	807.7088	813.6572
3	823.3496	819.4726
4	823.8627	822.1066
5	822.9925	822.6381
6	822.4515	822.5261
7	823.3838	823.0407

Y, por tanto, el valor que le corresponde a la hora 6 es, 823.0407 MBps.

3. ¿Qué técnica de predicción funciona mejor? ¿Por qué? ¿Cuál es la más adecuada para los datos con los que contamos?

Antes de decidir que técnica de predicción es la más adecuada, debemos poner en perspectiva los datos que estamos analizando. Actualmente, estamos calculando el valor que corresponde a la hora 6, el cual se predice mediante los valores anteriores. Estos valores anteriores realmente son medias de las decenas de miles de valores que teníamos previos a la realización de las medias. Por tanto, en este contexto, ahora mismo estamos trabajando con periodos de tiempo muy grandes (aunque sea de 1h). Esto se debe a que antes teníamos miles de valores incluso por cada milésima de segundo y ahora solo tenemos uno por hora.

Esto significa que anteriormente el periodo de tiempo entre muestra y muestra era ínfimo, y, por tanto, aplicar una predicción con medias móviles sería la opción teóricamente más acertada. Sin

embargo, ahora tenemos un periodo de tiempo de horas (miles de milésimas de segundo), es decir, un periodo exageradamente más grande que el anterior. En consecuencia, esto nos indica que la técnica de predicción más adecuada será el suavizado exponencial; que funciona mejor para periodos de tiempo grandes.

Claramente, si nos fijamos en los gráficos, podemos observar como lo comentado hasta ahora se refleja claramente. En particular, en el caso del tamaño, la diferencia entre medias móviles y suavizado exponencial es poca, pero existe. Sin embargo, en el caso de la velocidad, prácticamente es el mismo valor. Es por ello que esto puede llevarnos a la confusión, pero aplicando los conceptos teóricos vistos en clase, nos decantamos claramente por el suavizado exponencial en ambos casos.

Scripts

A continuación el script implementado:

```
import numpy as np
import statistics
import matplotlib.pyplot as plt

with open("data.arff", "r") as file:
    data = file.read().splitlines()

data = data[8:]

hours_label = ["22", "23", "01", "02", "03", "04", "05"]
pred_hours_label = ["22", "23", "01", "02", "03", "04", "05", "06"]
hours = np.arange(len(hours_label))


def size():
    # Plot initialization
    plt.figure()
    plt.rcParams.update({"font.family": "serif"})

    y_values = []

    # Get data for especific hour
    for hour in hours_label:
        y = get_axes_mean(hour, 0)
        plt.scatter(hour, y, s=10, c="blue")
        plt.xticks(hours, hours_label)

        # Data for post-processing
        y_values.append(y)

    # Save plot
    plt.title("Media de los Tamaños (MB) frente al Tiempo (h)")
    plt.xlabel('Tiempo (h)')
    plt.ylabel('Tamaño (MB)')
    plt.savefig(f"size.png")

    # Post-processing
    linear_regression(y_values, "Tamaño (MB)")
    moving_means(y_values, "Tamaño (MB)")
    exponential_smoothing(y_values, "Tamaño (MB)")
```

```
41
42
43  def velocity():
44      # Plot initialization
45      plt.figure()
46      plt.rcParams.update({"font.family": "serif"})
47
48      y_values = []
49
50      # Get data for especific hour
51      for hour in hours_label:
52          y = get_axes_mean(hour, 2)
53          plt.scatter(hour, y, s=10, c="blue")
54          plt.xticks(hours, hours_label)
55
56          # Data for post-processing
57          y_values.append(y)
58
59      # Save plot
60      plt.title("Media de las Velocidades (MBps) frente al Tiempo (h)")
61      plt.xlabel('Tiempo (h)')
62      plt.ylabel('Velocidad (MBps)')
63      plt.savefig(f"velocity.png")
64
65      # Post-processing
66      linear_regression(y_values, "Velocidad (MBps)")
67      moving_means(y_values, "Velocidad (MBps)")
68      exponential_smoothing(y_values, "Velocidad (MBps)")
69
70
71  def format(x):
72      return float(('%.4f' % x).rstrip('0').rstrip('.'))
73
74
75  def get_axes_mean(hour, value):
76      y = []
77
78      # Get data for especific hour
79      for line in data:
80          line = line.split(',')
81          if line[1] == hour:
82              y.append(float(line[value]))
83
84      # Calculate mean
```

```
    if value == 0:
        return format(np.mean(y))
    else:
        return format(statistics.harmonic_mean(y))


def linear_regression(y, type):
    x = np.arange(len(y))

    # Calculate means
    x_mean = np.mean(x)
    if type == "Velocidad (MBps)":
        y_mean = statistics.harmonic_mean(y)
    else:
        y_mean = np.mean(y)

    # Calculate b
    b = (np.sum(y * x) - len(y) * x_mean * y_mean) / (np.sum(x * x) - len(x) *
     ↪  x_mean * x_mean)

    # Calculate a
    a = y_mean - b * x_mean

    # Plot data
    plt.figure()
    plt.rcParams.update({"font.family": "serif"})
    plt.scatter(x, y, s=10, c="blue")

    # Plot linear regression
    x_plot = np.arange(len(pred_hours_label))
    plt.plot(x_plot, a + b * x_plot, color='red')

    # Save plot
    plt.xticks(x_plot, pred_hours_label)
    plt.title(f"Regresión Lineal de {type}")
    plt.xlabel('Tiempo (h)')
    plt.ylabel(f"{type}")
    plt.savefig(f"linear_regression_{type}.png")


def moving_means(y, type):
    i = 1
    moving_averages = []
    cum_sum = np.cumsum(y)
```

```
    x = np.arange(len(y))

    # Calculate moving averages
    i = 4
    while i <= (len(y)):
        window_average = round(cum_sum[i-1] / i, 2)
        moving_averages.append(window_average)
        i += 1

    # Calulate moving averages for next hour
    moving_averages.append(np.sum(y[-4:])/4)

    # Plot data
    plt.figure()
    plt.rcParams.update({"font.family": "serif"})
    plt.scatter(x, y, s=10, c="blue")

    # Plot moving averages
    x_plot = np.arange(3, 8)
    plt.plot(x_plot, moving_averages, color="red")

    # Save plot
    x = np.arange(len(pred_hours_label))
    plt.xticks(x, pred_hours_label)
    plt.title(f"Media Móvil de {type}")
    plt.xlabel('Tiempo (h)')
    plt.ylabel(f"{type}")
    plt.savefig(f"moving_means_{type}.png")


def exponential_smoothing(y, type):
    x = np.arange(len(y))

    # Plot data
    plt.figure()
    plt.rcParams.update({"font.family": "serif"})
    plt.scatter(x, y, s=10, c="blue")

    # Exponential smoothing
    alpha = 0.6
    smoothed = [y[0]]
    for i in range(1, len(y)):
        smoothed.append(alpha * y[i] + (1 - alpha) * smoothed[i-1])

```

```
    # Calculate moving averages for next hour
    smoothed.append(smoothed[-1])

    # Plot exponential smoothing
    x_plot = np.arange(8)
    plt.plot(x_plot, smoothed, color="red")

    # Save plot
    plt.xticks(x_plot, pred_hours_label)
    plt.title(f"Suavizado Exponencial de {type}")
    plt.xlabel('Tiempo (h)')
    plt.ylabel(f"{type}")
    plt.savefig(f"exponential_smoothing_{type}.png")


if __name__ == "__main__":
    size()
    velocity()
```