Universidad de Valladolid
E.T.S Ingeniería Informática
Grado en Ingeniería Informática [Tecnologías de la información]

Curso 2017/2018
Evaluación de Sistemas Informaticos
Práctica de Laboratorio 1
Evaluación del rendimiento de un servidor web.

Grupo de Laboratorio 07
Esteban Pellejero, Sergio
González Bravo, Miguel
González Pérez, Daniel
Rabadán Martín, Borja

Índice

1. Introducción

En esta primera práctica se va a evaluar la productividad de un servidor web Apache instalado en las máquinas virtuales proporcionadas por la escuela. El objetivo es dar una aproximación de los niveles de servicio que ofrece, y de esta manera poder ajustarlo para que funcione de forma óptima. Por otro lado, el estudio servirá para predecir nuevos valores futuros, lo que permite, entre otras cosas, conocer cómo se va a comportar nuestro servidor con periodos de carga en situaciones de estrés y congestión. Nuestro informe se basará en el estudio de los objetivos generales de la evaluación del rendimiento en un sistema informático, que son:

- Detección de cuellos de botella o fallos en el sistema. Ajustes del sistema.
- El análisis y análisis comparativo de sistemas o aplicaciones.
- Comparación de diferentes características de rendimiento.
- Identificación de niveles de productividad de un sistema.

Como se ha remarcado, se va a realizar un intento de reflejar estas actividades centrando la atención en nuestro servidor web a lo largo de este documento.

2. Sistema en evaluación

En esta sección viene recogida información hardware y software del servidor a analizar.

2.1. Hardware

Las características del servidor, obtenidas mediante comandos, son las siguientes:

- **Procesador**: A continuación se muestra información acerca de la marca, la cantidad de núcleos y la velocidad en MHz, analizando el fichero */proc/cpunifo* de la siguiente manera:

```
$ grep "vendor_id" /proc/cpuinfo ; grep "cpu cores" /proc/cpunifo ; grep
    "cpu MHz" /proc/cpuinfo
```

  Que devuelve la siguiente salida:

```
$ vendor_id                     : GenuineIntel
$ cpu cores                     : 2
$ cpu MHz                       : 1995.192
```

  Conocer, por ejemplo, la velocidad y la cantidad de núcleos de los que se dispone puede ser relevante para el tener una aproximación de la concurrencia real de nuestro sistema a la hora de aceptar clientes por parte del servidor web.

- **Memoria**: A continuación se muestra información acerca de la cantidad de memoria RAM junto con la de intercambio (swap) disponible, mediante el comando free:

```
$ free
```

Que devuelve la siguiente salida:

```
$            total     used     free    shared  buff/cache   available
$ Memoria:   1860984   219996   750244  26596   890744       1444024
$ Swap:      131068    21552    109516
```

Que, de forma simplificada, quiere decir que se dispone de $\simeq$ 1.91 Gb de memoria RAM y $\simeq$ 131 Mb de memoria de intercambio.

- **Almacenamiento secundario**: A continuación se muestra información acerca de del los discos de almacenamiento masivo, a través del comando fdisk -l (en modo root):

```
$ fdisk -l
```

Que devuelve lo siguiente:

```
$ Disk /dev/vda: 6 GiB, 6442450944 bytes, 12582912 sectors
$ Units: sectors of 1 * 512 = 512 bytes
$ Sector size (logical/physical): 512 bytes / 512 bytes
$ I/O size (minimum/optimal): 512 bytes / 512 bytes
$ Disklabel type: dos
$ Disk identifier: 0x459fa249
$
$ Disposit.  Inicio Start    Final Sectores Size Id Tipo
$ /dev/vda1          2048 12582911 12580864   6G 83 Linux
```

En resumen, se dispone de un disco duro de 6 Gb, que es más que suficiente para alojar el servidor web.

2.2. Información software:

A continuación se muestra información acerca del sistema operativo, junto con el servidor Apache para lanzamiento y recogida de pruebas instalado con los comandos utilizados.

- **Sistema Operativo**: Información obtenida a través del fichero */etc/os-release* con el comando:

```
$ cat /etc/os-release
```

Que entre otra información nos devuelve:

```
$ NAME="Ubuntu"
$ VERSION="16.04.3 LTS (Xenial Xerus)"
```

- **Versión de Apache**: En concreto, nos interesa conocer la versión del servidor Apache disponible:

```
$ apache2 -v
```

Que nos devuelve la siguiente información:

```
$ Server version: Apache/2.4.18 (Ubuntu)
```

A parte de conocer la versión también se debe comprobar que el servidor tenga la opción de Keep-alive habilitada. Esta función sirve para que las conexiones sean persistentes, es decir, que pueda haber más de una solicitud al servidor por conexión, lo cual va a ser necesario para nuestro estudio. Para ello, se modifica la línea 92 del archivo */etc/apache2/apache2.conf* y se escribe lo siguiente:

```
$ KeepAlive On
```

Por último, destacar que las características sobre la red no van a ser analizadas. El servidor web en cuestión es una máquina virtual situada en los servidores de la escuela. Hay que tener en cuenta que si las pruebas de rendimiento se lanzan dentro de la red interna de la escuela desde máquinas muy próximas, la conexión con el servidor va a ser más rápida que si se establece desde fuera, pues no hay interferencias posibles que afecten al rendimiento del servidor. Por lo tanto, en este estudio no se va a tener en cuenta la influencia del factor red. Todas las pruebas han sido lanzadas desde la red interna de la escuela.

3. Planificación y diseño de las pruebas

En esta sección se hablará de los patrones de utilización y diseño de los escenarios de utilización. En nuestro caso las condiciones del experimento y el conjunto de pruebas a ejercitar sobre el sistema ya nos vienen dadas según la **Figura 1**.

		Número de peticiones				
		1000	5000	10000	50000	100000
	50	IC1				
Clientes	100	IC2	IC3	IC5	IC7	IC10
Concurrentes	250		IC4	IC6	IC8	IC11
	500				IC9	IC12

Figura 1: Intensidades de carga y niveles de concurrencia

3.1. Condiciones de ejecución del experimento

En nuestro caso, las condiciones de ejecución del experimento vienen indicadas en el enunciado de la práctica (si no dispusiésemos ese tipo de información, se debería de crear un modelo de carga característico y diseñado específicamente para nuestro servidor). A partir de ahí. vamos a utilizar una de las herramientas de benchmarking disponibles para el lanzamiento de pruebas, en nuestro caso ab (*Apache benchmark*). Las condiciones impuestas para ello son:

- La ejecución de los test se ejecutará con la máquina dedicada para evitar interferencias.

- Todos los test se ejecutarán desde el servidor Apache en modo keep-alive.
- El servidor se someterá a las intensidades de carga indicadas en la **Figura 1**.
- Para cada nivel de carga, cada caso de prueba se ejecutará al menos 5 veces.

3.2. Casos de prueba

En esta práctica se ha estimado correcto trabajar con 2 tipos de cargas diferentes. Para ello, se van a realizar pruebas sobre páginas web estáticas y dinámicas que simulen este comportamiento. Las páginas web estáticas son las que el servidor simplemente devuelve el documento HTML, sin realizar más operaciones. Las páginas web dinámicas son las que ejecutan algún tipo de programa en la parte del servidor, por ejemplo PHP o JavaScript. En nuestro caso las diferentes páginas que hemos puesto en el servidor tienen las siguientes características:

- **Páginas web estáticas**:
  - *Página web "pequeña"*: El contenido HTML5 es menor de 50 Kb:

```
$ <html>
$   <head>
$     <meta http-equiv="Content-Type" content="text/html;charset=utf
    -8"/>
$     <title>ESI</title>
$   </head>
$   <body>
$     <H1>Evaluacion y Rendimiento de Sistemas Informaticos</H1>
$         <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
     sed do eiusmod tempor incididunt ut labore et dolore magna
    aliqua. Ut $            enim ad minim veniam, quis nostrud
    exercitation ullamco laboris nisi ut aliquip ex ea commodo
    consequat. Duis aute irure dolor $               in
    reprehenderit in voluptate velit esse cillum dolore eu fugiat
    nulla pariatur. Excepteur sint occaecat cupidatat non        $
                    proident, sunt in culpa qui officia deserunt
    mollit anim id est laborum. </p>
$     <img src="pequena.jpg">
$   </body>
$ </html>
```

  - *Página web "grande"*: El contenido HTML5 es mayor de 250 Kb:

```
$ <html>
$   <head>
$     <meta http-equiv="Content-Type" content="text/html;charset=utf
    -8"/>
$     <title>ESI</title>
$   </head>
$   <body>
$     <H1>Evaluacion y Rendimiento de Sistemas Informaticos</H1>
$     <img src="grande.jpg">
$   </body>
$ </html>
```

- **Página web dinámica**: Contiene un ligero programa que conlleva una ejecución en el servidor, por lo que consume recursos del mismo, y se supone que va a generar más carga que

una simple web estática. En nuestro caso, simplemente se ejecuta la función phpinfo(), que muestra gran cantidad de información acerca de este lenguaje de programación:

```
$ <html>
$   <head>
$     <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
$     <title>ESI</title>
$   </head>
$   <body>
$     <H1>Evaluacion y Rendimiento de Sistemas Informaticos</H1>
$     <img src="pequena.jpg">
$     <?php phpinfo() ?>
$   </body>
$ </html>
```

Estas páginas web han sido introducidas en el servidor. Para ello, simplemente se han generado los documentos correspondientes en la carpeta del sistema */var/www/html/*, donde se sitúan los 3 documentos llamados *webPequena.html*, *webGrande.html* y *hola.php*. A continuación se muestra el resultado de acceder a una de las páginas web, a modo de ejemplo y como comprobación de su funcionamiento:

Figura 2: Correcto funcionamiento de la página web pequeña

4. Preparación del entorno de pruebas

En los servidores web, las medidas necesarias para parametrizar el modelo de rendimiento se reducen a conocer las demandas de procesador, la E/S y y la red que establecen las peticiones

HTTP; siendo estos los dispositivos hardware que más intervienen en la consecución del proceso solicitado. Los valores de dichas demandas específicas para el modelo de carga del servidor web se obtendrán de experimentos controlados de benchmarking y su posterior análisis operacional.

Un modo muy útil para estudiar el rendimiento de un sistema es someterlo a una parte de la carga actual y medir los valores de interés del mismo. En contrapartida, la carga real es difícil de medir, por lo que en la mayoría de los casos no suele ser efectivo.

La alternativa general es hacer uso de los programas de pruebas (*benchmark*) y utilizar sus resultados para comprender el rendimiento de los diferentes sistemas. Para ello, vamos a utilizar el programa ab, ya que este es una herramienta parametrizable. Nuestro comando de ab que ejecutará las pruebas es el siguiente (a modo de ejemplo):

```
$ ab -k -c 100 -n 50000 -g ./informes/ic7/graficos$2$1.tsv -r -s 50 http:
    //virtual.lab.inf.uva.es:31072/$2 > ./informes/ic7/datos$2$1
```

Los parámetros asignados significan lo siguiente:

- **k**: Indica que queremos activar el modo keep-alive, esto quiere decir que mantenemos los canales de comunicaciones abiertos. De la otra manera, cada vez que se hace una petición HTML se abre un canal por cada objeto que se pide (imágenes, servlets, etc.)
- **c**: Indica el nivel de concurrencia, número de peticiones simultáneas ejecutadas en un determinado periodo de tiempo.
- **n**: Indica el número de peticiones a realizar en una sesión de benchamrking.
- **g**: Escribe toda la salida de los resultados en un formato de archivos entendible para el programa de representaciones gráficas gnuplot.
- **r**: Indica al programa que no tiene que cancelar la ejecución cuando el socket genera algún error de comunicaciones.
- **s**: Número máximo de segundos a esperar después de que el socket genere un time out.

Los ficheros obtenidos de las pruebas se han obtenidode recoger la información de los casos de prueba tanto en formatos gráficos como las propias salidas por pantalla del propio comando. Cada una de ellas ha sido las almacenada en su fichero correspondiente, indicando el número de la intensidad de carga (ic) y la página web que estamos pidiendo al servidor. A partir de ahí, posteriormente se generará un pequeño programa que tome los datos necesarios de manera automática para su posterior análisis.

La salida típica de una ejecución del comando anterior, es la que se muestra a continuación. Hay que entenderla para poder analizar posteriormente los datos de manera correcta:

```
$ This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
$ Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net
    /
$ Licensed to The Apache Software Foundation, http://www.apache.org/
$
$ Benchmarking virtual.lab.inf.uva.es (be patient)
$
```

```
$
$ Server Software:        Apache/2.4.18
$ Server Hostname:        virtual.lab.inf.uva.es
$ Server Port:            31072
$
$ Document Path:          /webPequena.html
$ Document Length:        241 bytes
$
$ Concurrency Level:      50
$ Time taken for tests:   5.330 seconds
$ Complete requests:      1000
$ Failed requests:        0
$ Total transferred:      511000 bytes
$ HTML transferred:       241000 bytes
$ Requests per second:    187.61 [#/sec] (mean)
$ Time per request:       266.504 [ms] (mean)
$ Time per request:       5.330 [ms] (mean, across all concurrent
    requests)
$ Transfer rate:          93.62 [Kbytes/sec] received
$
$ Connection Times (ms)
$               min  mean[+/-sd] median   max
$ Connect:        1  147 461.8      1    3048
$ Processing:     1   27  78.6      1     832
$ Waiting:        1   27  78.6      1     832
$ Total:          2  174 491.8      2    3256
$
$ Percentage of the requests served within a certain time (ms)
$  50%      2
$  66%      3
$  75%      3
$  80%      5
$  90%   1009
$  95%   1033
$  98%   1370
$  99%   3247
$ 100%   3256 (longest request)
```

El objetivo es analizar tres características del servidor: el rendimiento del servidor web en base a su comportamiento (en el caso de que estemos utilizando el monitor de rendimiento *SAR*), los tiempos de respuesta del mismo y su productividad en función de la carga a la que se ve sometido. Una vez tenemos claro los objetivos de nuestro análisis, hay que asociarlos con las correspondientes medidas que nos devuelven las ejecuciones del ab:

- **Rendimiento del servidor**: Para realizar esta tarea se ha utilizado en el servidor el paquete de herramientas de sysstat, de las cuales se van a utilizar en concreto SADC y SAR. El primero se encarga de recoger los datos relevantes del sistema y almacenarlos en ficheros binarios de tipo logs, mientras que el segundo se encarga de traducir esos ficheros de logs binarios a ficheros legibles para los seres humanos.

  Se ha decidido que la manera de llevar a cabo la monitorización del servidor a través de estas herramientas va a ser mediante la ejecución del SADC, pero viendo los resultados con el SAR para cada conjunto de pruebas ejecutadas sobre el servidor.
  Por ejemplo, ejecutando el SADC antes de lanzar peticiones con carga sobre la página web pequeña. Es cierto que así tenemos una visión más global porque por ejemplo vemos que conforme las intensidades de carga van aumentando, seguramente la demanda de recursos del servidor será mayor. Pero tenemos la ventaja de que no medimos individualidades y

picos de carga que captaríamos si ejecutásemos el SADC durante cada intensidad de carga.

- **Tiempos de respuesta**: para esto usaremos los valores de *time taken for request*, que nos da la duración global de las pruebas y los *time per request* que nos devuelve la media del tiempo que tarda en procesarse de manera satisfactoria una petición HTTP.
- **Productividad**: La productividad hace referencia a la cantidad de trabajo útil realizado por unidad de tiempo en un entorno de carga determinado [RAJ JAIN]. Normalmente la productividad es medida en operaciones por segundo y proporciona un índice de la velocidad de ejecución de un conjunto de programas (denominado carga de trabajo). Para medir el grado de productividad que tiene nuestro servidor se van a utilizar dos medidas diferentes:
  - *Request per second*: que es la productividad, número de peticiones por segundo.
  - *Transfer rate*: productividad medida en KB por segundo.

Para facilitar este trabajo se ha realizado un script que recoge todas las pruebas a lanzar, que comprende los diferentes modelos de carga e intensidades. El script se muestra en el ANEXO I.

El funcionamiento básico del script es el siguiente. Primero lanza las diferentes pruebas sobre las páginas web, con diferentes intensidades de carga. Al programa se le pasa como parámetro la página web que queremos probar y el número de la prueba (hay que hacer 5 pruebas a diferentes horas del día para cada página web).

El programa inicia la monitorización en el servidor (SAR), y comienza a lanzar las pruebas. Deja un lapso de tiempo de 10 segundos entre intervalos de carga para que el servidor se descargue un poco.

Una vez terminado el script, tenemos las carpetas ./informes/ic[1-12], donde podemos encontrar los datos generados por el Apache Benchmark. Ahora comentaremos como hemos procesado los datos y los programas utilizados.

5. Análisis exploratorio de datos

Para el análisis exploratorio de datos hemos decidido usar dos programas muy sencillos que procesan los datos. Estos programas son propios, ya que las operaciones que se realizan son muy básicas (medias, desviaciones típicas, intervalos de confianza, etc).

El primer script que ejecutamos recoge los datos proporcionados por el Apache benchmark y los deja en un formato legible para su posterior análisis completo. El programa en cuestión es el siguiente:

```
#!/bin/bash

#Args:
#$1: Filename

#Lo que dura la prueba (s)
grep "Time taken for tests" $1|cut -d " " -f7 > $1

#Requests fallidos (Fiabilidad)
```

```
grep "Failed requests" $1|cut -d " " -f10 >> $1

#Request por segundo (Productividad)
grep "Requests per second" $1|cut -d " " -f7 >> $1

#Tiempo por request (ms) (tiempos respuesta)
grep "across all concurrent" $1|cut -d " " -f10 >> $1
```

Con este programa, procesamos los ficheros generados directamente por el AB y extraemos la información numérica que nosotros deseamos y la almacenamos en ficheros temporales:

- **Time taken for test**: duración completa de cada intervalo de carga.
- **Failed request**: peticiones fallidas, con esto podemos medir la fiabilidad del servidor.
- **Request per second**: peticiones por segundo, para medir la productividad.
- **Time per request**: tiempo de respuesta del servidor para cada una de las peticiones, sirve para ver los tiempos de respuesta del servidor.

Aunque datos como las peticiones fallidas y el tiempo total de las pruebas no son el objetivo principal de esta práctica, pueden dar una buena idea del funcionamiento del servidor.

Aunque antes hayamos dicho que la productividad también la íbamos a medir según el parámetro *Transfer rate*, que es la productividad medida en KB por segundo, es absurdo hacer esto ya que al medirla mediante las peticiones, ganamos un nivel de abstracción mayor y es más entendible.

Ahora para calcular los correspondientes datos de medias, intervalos de confianza y desviaciones, mostrados en las posteriores hojas y gráficas, se calculan con un programa en Java que podemos encontrar en el ANEXO II.

El funcionamiento de este programa es muy básico. Primero coge los datos de los archivos temporales generados por el script anterior. Luego calculamos las medias y desviaciones estándar de cada uno de los diferentes atributos recogidos. Esto lo hacemos para cada uno de los intervalos de carga en las diferentes pruebas. Con estos datos, y otros estándares (datos relacionados con la distribución normal, t de student, etc.) podemos calcular el intervalo de confianza al 95 % para estos datos. La fórmula para obtener el intervalo de confianza es la siguiente:

$$[\bar{x} - t_{1-\frac{\alpha}{2},n-1} \cdot \frac{\sigma}{\sqrt{n}}, \bar{x} + t_{1-\frac{\alpha}{2},n-1} \cdot \frac{\sigma}{\sqrt{n}}]$$

Explicación de los datos:

- $\bar{x}$: media obtenida a partir de los datos con el programa Java.
- σ: desviación estándar, obtenida a partir de los datos con el programa Java.
- **n**: 5, que son la cantidad de pruebas que hemos realizado, que al ser menor de 30, tenemos que suar la t de Stundet, sino, tendríamos que realizar los cálculos con Z, que sería una distribución normal.
- $t_{1-\frac{\alpha}{2},n-1}$: simplemente tenemos que coger una de las tablas estadísticas usadas para estos casos y mirar su valor, que en este caso es aproximadamente $\simeq 2.7765$.

Para poder rellenar estas tablas, como el archivo de java solo saca los datos para un IC, hemos hecho un script que lo calcula y saca por pantalla la información de todos los intervalos de carga. Este programa se puede ver en el ANEXO III. Simplemete va recorriendo todos los intervalos de carga y calculando las medias de las 5 pruebas para cada documento, luego muestra la información por pantalla en formato legible para programas de procesado de gráficas, que en nuestro caso hemos usado una hoja de cálculo ya que la cantidad de datos no era muy grande.

Las tablas con las medias son las siguientes:

	IC1	IC2	IC3	IC4	IC5	IC6	IC7	IC8	IC9	IC10	IC11	IC12
Tiempo total (seg)	5,001	5,02	8,023	12	14,394	16,882	86,759	97,172	73,238	143,391	151,453	172,107
Fallos	0	0,2	1,2	41,6	0,4	50	9,2	155,2	159	23,4	104,4	589,6
Productividad (pet/seg)	204,908	243,276	638,358	444,01	734,996	641,058	689,814	631,19	725,494	772,574	729,874	697,464
Velocidad Respuesta (seg)	5,001	5,02	1,605	2,4	1,439	1,688	1,735	1,943	1,465	1,434	1,515	1,721

Figura 3: Medias de las pruebas Web Pequeña

	IC1	IC2	IC3	IC4	IC5	IC6	IC7	IC8	IC9	IC10	IC11	IC12
Tiempo total (seg)	3,743	4,482	6,111	9,738	13,243	15,025	64,843	66,286	67,439	155,968	163,052	148,377
Fallos	0	0,2	1,8	47	4,6	42,6	12,8	75	196,4	68,2	182,6	473,4
Productividad (pet/seg)	373,914	259,984	827,894	549,422	758,792	683,068	803,708	787,574	772,662	742,662	728,1	750,422
Velocidad Respuesta (seg)	3,743	4,482	1,222	1,947	1,324	1,502	1,297	1,326	1,349	1,56	1,63	1,484

Figura 4: Medias de las pruebas Web Grande

	IC1	IC2	IC3	IC4	IC5	IC6	IC7	IC8	IC9	IC10	IC11	IC12
Tiempo total (seg)	14,128	14,052	113,575	122,588	244,315	208,336	1206,746	1162,072	1163,815	1631,713	2013,314	2245,874
Fallos	101	112	500	486	991	1196	5728	9032	13395	15124	21405	27048
Productividad (pet/seg)	70,78	71,16	44,02	40,79	40,93	48	41,43	43,03	42,96	43,74	44,83	44,53
Velocidad Respuesta (seg)	14,128	14,052	22,715	24,518	24,431	20,834	24,135	23,241	23,276	23,358	22,845	22,459

Figura 5: Medias de las pruebas Web Dinámica

Una vez que tenemos las tablas rellenadas, podemos extraer unas pequeñas gráficas de los datos requeridos para poder comprender mejor los resultados. En principio hemos hecho gráficas para los objetivos principales de esta práctica, que son medir la productividad y el tiempo de respuesta del sistema. Esta última característica la hemos separado en dos gráficas debido a las diferentes magnitudes al haber hecho las pruebas, por una parte mostramos el tiempo de respuesta de del servidor al servir páginas dinámicas y por otra las estáticas La diferencia entre ambos tipos es bastante grande. También hemos añadido una pequeña explicación para las otras dos métricas, no tan extensas como las principales.

Primero haremos unos comentarios a cerca de las gráficas obtenidas y en el punto 6, hablaremos de las conclusiones y porqué se producen esos resultados.

5.1. Productividad del sistema

Como hemos dicho la productividad de un sistema es la cantidad de trabajo útil ejecutado por unidad de tiempo en un entorno de carga determinado. Nosotros lo hemos medido en peticiones por segundo a un servidor web. En nuestro caso se puede ver reflejada en la **Figura 6**

Una de las primeras impresiones es que la diferencia gráfica de la productividad entre las páginas de tipo estático y dinámico. La productividad para las páginas dinámicas es nefasta, es casi una recta de valores entre 40 y 70.

Otra de las características es que la productividad para las páginas estáticas no varía demasiado, siendo la página web grande la que tiene mayor productividad durante casi toda la duración de las pruebas.

Para el análisis de las páginas estáticas podemos usar el modelo del *knee load*, que evidentemente no es tan perfecto como el visto en teoría, pero si que se le asemeja. Podemos observar que la productividad crece de una manera rápida hasta las intensidades de carga 3-5, donde se encontraría nuestro *knee load* y a partir de esos intervalos de carga se estabiliza de una manera más o menos regular, es decir, no sube ni baja de manera tan notoria. A la vista de los resultados, según el modelo del *knee load*, nuestra productividad máxima estaría entre las intensidades de carga 9 y 10.

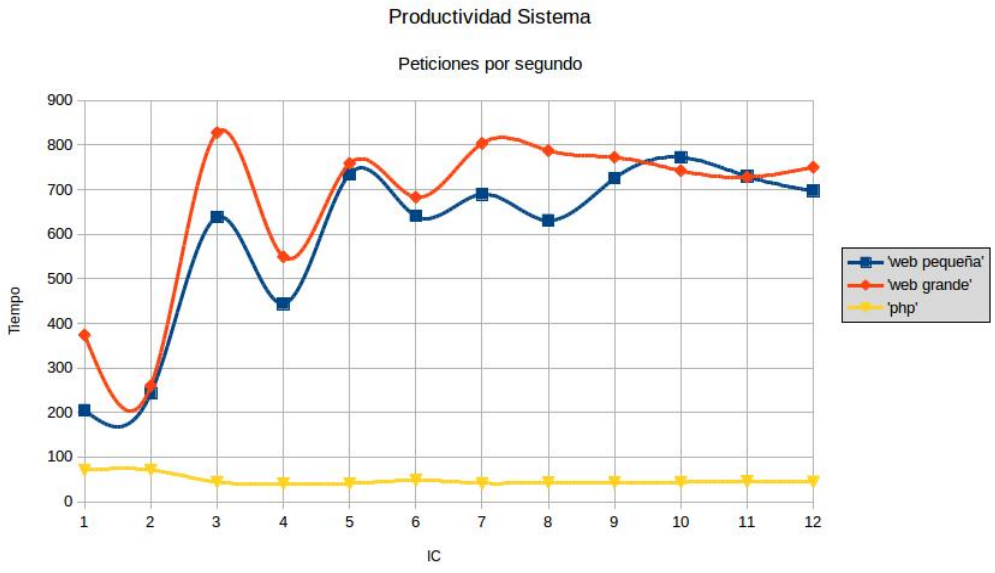

Figura 6: Gráfica de la productividad del sistema

5.2. Tiempo de respuesta del sistema

El tiempo de respuesta en nuestro sistema web es el tiempo medio que tardaba el servidor en servir las diferentes páginas web, medido en segundos.

Al igual que en el caso de la productividad, la primera diferencia grande se produce entre las páginas web estáticas (**Figura 8**) y las dinámicas (**Figura 7**). La página web dinámica tiene unos tiempos de respuesta relativamente altos, se tarda en servir cada una de las peticiones entre 20 y 25 segundos, lo cual es mucho tiempo. Al principio podemos observar que se tarda menos, pero a partir del IC3, que ya tiene una carga considerable, el tiempo de respuesta se dispara.

En las páginas estáticas observamos que al principio el tiempo de respuesta de las primeras intensidades de carga es elevado (5 segundos) mientras que luego baja considerablemente y se estabiliza entorno a los 1.5 - 2 segundos. En el apartado de conclusiones veremos porqué se produce esto.

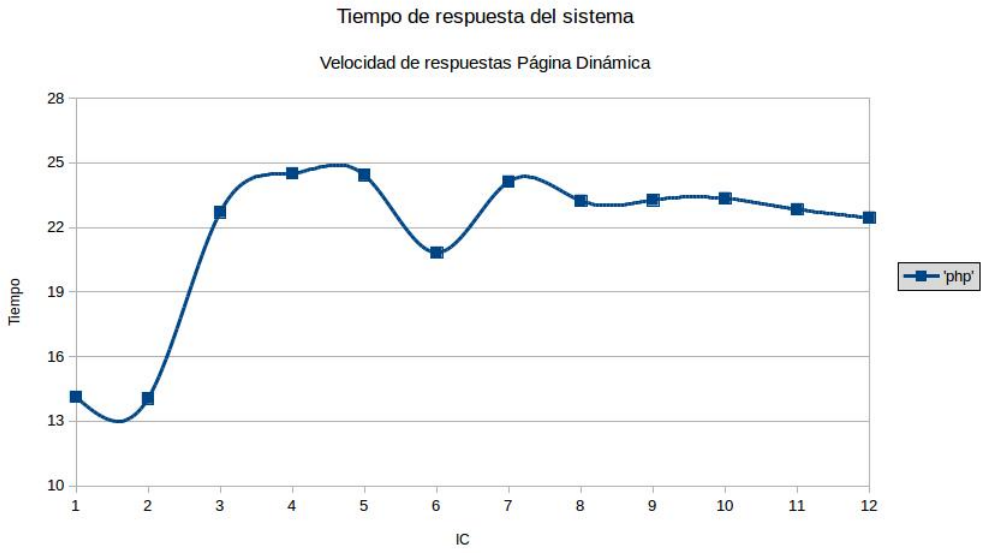

Figura 7: Gráfica del tiempo de respuesta para páginas dinámicas

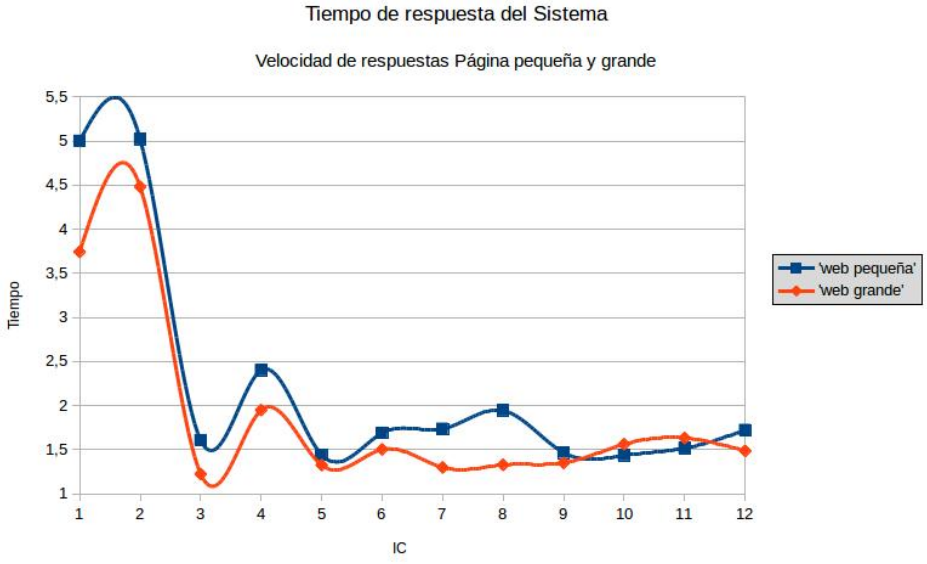

Figura 8: Gráfica del tiempo de respuesta para páginas estáticas

5.3. Tiempos de ejecución de pruebas

Sobre los tiempos de ejecución de las pruebas no vamos a entrar en demasiado detalle. Las gráficas de los tiempos de ejecución de las pruebas en las páginas estáticas (**Figura 9**) y dinámicas (**Figura 10**) están separadas porque juntas, al tener escalas diferentes no eran representativas.

Lo importante es que todas las páginas siguen una distribución que se aproxima a la exponencial, es decir, los tiempos de ejecución de las pruebas aumentan según una ecuación exponencial (aproximadamente). En las conclusión diremos el porqué de esto, ya que es bastante lógico y un resultado esperado, quizá no tan esperado que sea exponencial, pero si que a mayor IC mayor sea el tiempo de espera.

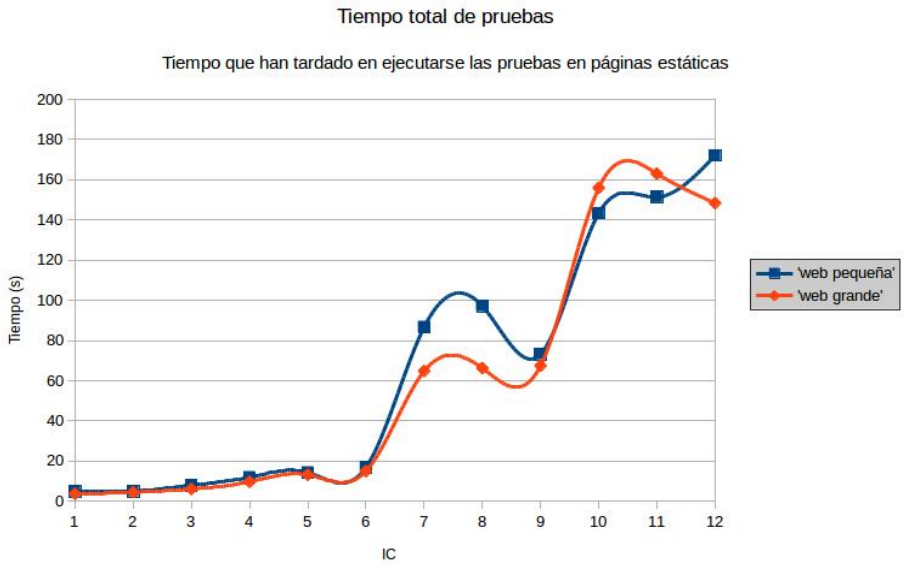

Figura 9: Gráfica del tiempo de ejecución de pruebas para páginas estáticas

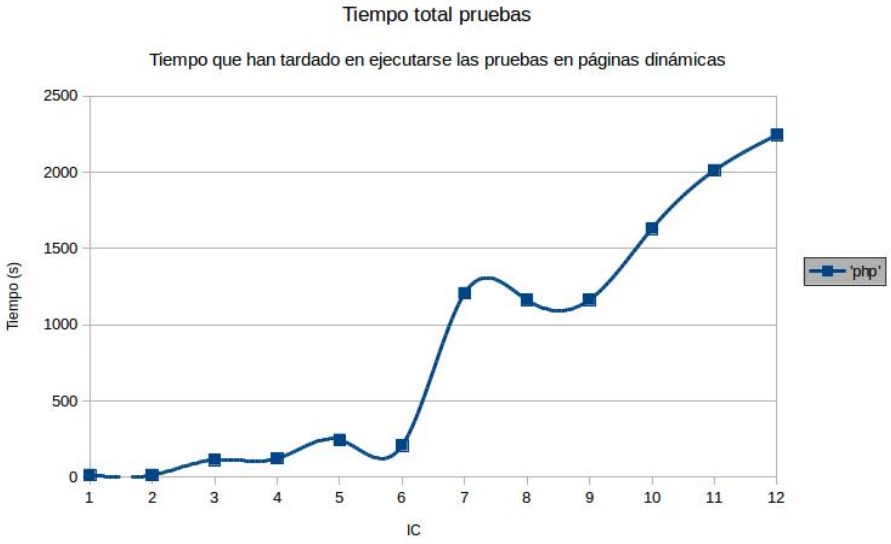

Figura 10: Gráfica del tiempo de ejecución de pruebas para páginas dinámicas

5.4. Fiabilidad

La fiabilidad del sistema. Con esta medida podemos saber con una cierta precisión la fiabilidad que va a tener nuestro sistema bajo unas ciertas condiciones de carga del entorno. En nuestro caso la fiabilidad se basa en la cantidad de peticiones que se le hacen al servidor en relación con las que no se han completado de manera correcta.

Al igual que en el caso anterior, las gráficas entre las páginas estáticas (**Figura 11**) y dinámicas (**Figura 12**) están separadas por temas de escala. Algo curioso es que las páginas estáticas siguen una distribución de fallos que se aproxima más a una distribución exponencial. Mientras que en las páginas dinámicas, los fallos hasta el IC6 son más o menos constantes y muy poco apreciables, sin embargo a partir del IC6 crecen de manera lineal.

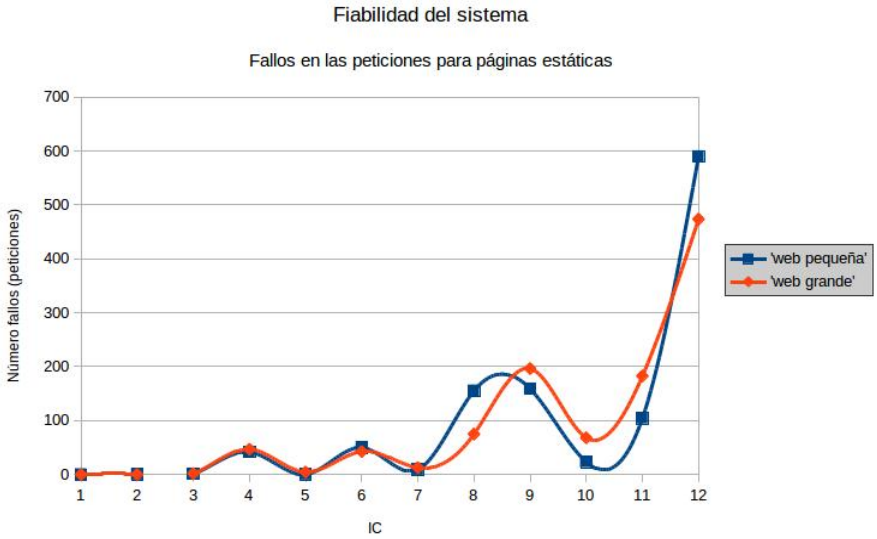

Figura 11: Gráfica de fallos en peticiones para páginas estáticas

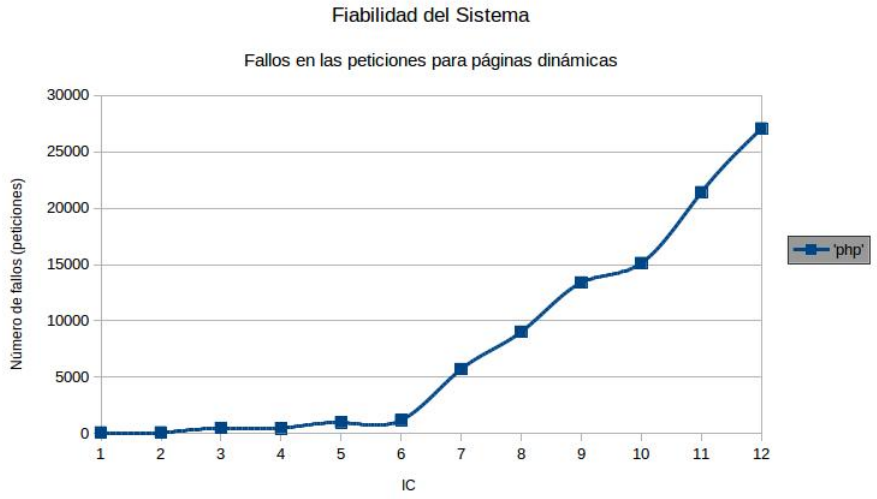

Figura 12: Gráfica de fallos en peticiones para páginas dinámicas

6. Conclusiones

Una vez que hemos observado los resultados obtenidos en forma de gráfica, y hemos hablado un poco de como se comportan las diferentes variables en función de la carga del sistema, vamos explicar porqué se dan estos comportamientos.

6.1. Productividad y tiempos de respuesta

Estas dos características están muy ligadas entre sí, así que las vamos a explicar de una manera mas o menos paralela.

Recordamos que las páginas estáticas tienen una mayor productividad que las dinámicas. Esto se debe a que las páginas estáticas cuentan con ventaja, simplemente realizan una petición al servidor y este le manda los archivos, además contamos con la opción keep alive en el servidor que mantiene las conexiones TCP abiertas para que la productividad de la red se aproveche mejor.

Otra de las ventajas de las páginas estáticas es la existencia de memorias caché. Además en este entorno de pruebas jugamos con varios niveles de memorias caché. Los ordenadores suelen tener una pequeña cache de archivos, por lo que las primeras veces el servidor se trae el archivo que hay que enviar de disco, pero las siguientes peticiones lo tiene guardado en la memoria RAM.

Otro nivel de abstracción es la caché de las máquinas virtuales (hipervisor) si detecta que una máquina virtual hace muchas peticiones a un página web seguramente la deje almacenada en la memoria del servidor de las MV. Si las pruebas están realizadas desde casa también entra en juego el servidor web caché que tiene la escuela, que disminuye el tiempo que se tarda en servir las páginas web y por lo tanto aumenta la productividad.

Sin embargo en las páginas dinámicas, como nuestro phpinfo(), no se puede hacer eso, ya que la petición le llega al servidor, este tiene que generar toda la salida consumiendo los recursos del servidor y enviar la respuesta al cliente, es por esto que los tiempo de respuesta para las páginas dinámicas son mayores y por lo tanto la productividad del servidor para servir las páginas dinámicas es bastante mala.

Por lo tanto tenemos que tener cuidado al incluir contenido dinámico en nuestras páginas web. Por ejemplo si nuestro servicio web está orientado de tal manera que en ciertas fechas hay mucha intensidad de carga, es mejor no usar tanto contenido dinámico.

Sin embargo si nuestro portal web, se mantiene con unos acceso mas o menos regulares a lo largo del año, podemos meter más contenido dinámico a la página sin miedo a que afecte demasiado a la experiencia del usuario.

Ahora vamos a pasar a explicar sin entrar en profundidad las otras dos métricas utilizadas, los tiempos de ejecución de las pruebas y la fiabilidad del sistema.

6.2. Tiempos ejecución

Como vimos en el punto 4, los tiempos de ejecución de las pruebas aumentaban conforme se aumentaba la intensidad de carga. Esto es lógico debido a que cuanto mayor es la Intensidad de Carga, las peticiones se hace con un nivel de concurrencia mayor y también se realiza un número mayor de peticiones. Al principio los tiempos de ejecución son pequeños porque se realizan pocas peticiones pero conforme van aumentando y se van agotando los recursos de la máquina, este tiempo de ejecución va en aumento.

En resumen, cuanto mayor sea la carga que soporta nuestro servidor mayor serán los tiempos de ejecución de las pruebas.

6.3. Fiabilidad del sistema

A la vista de los resultados podemos decir que nuestro sistema tiene mucho mayor grado de fiabilidad cuanto menor es la carga del mismo. Estos resultados son los esperados debido a que conforme se va realizando un número mayor de peticiones, los recursos del sistema se van agotando.

Por ejemplo, imaginemos que entran al sistema 500 clientes a la vez, y el sistema tiene que lanzar 500 hilos para atender a cada cliente esto supone un gasto de memoria y cpu. Un aumento en la memoria puede llevar a la saturación de la misma, y según el sistema operativo que se esté ejecutando algunas soluciones son, por ejemplo, empezar a matar procesos (hilos) para poder liberar la memoria (y cpu), lo que conlleva a peticiones fallidas en el sistema. Este tipo de efectos son los que seguramente se hayan reflejado en nuestras gráficas de fiabilidad.

7. Bibliografía

[1] J.M. Marqués. Transparencias de la asignatura Evaluación de Sistemas Informáticos.

ANEXO I.

Script de lanzamiento de las pruebas de apache benchmark:

```
#!/bin/bash

# comandos de AB:
#  -c: concurrency, number of multiple request to perform at a time.
#  -n: number of requests to perform for the benchmarking.
#  -g: gnuplot for the statics.
#  -r: no sale dle programa cuando recibe errores de sockets.
#  -s: timeout for the messages, 50 seconds, if the server has much load.
#      tiene un timeout de 1 min, -s 60 (por defecto son solo 30 seg)
timeOut=60

# $1: el numero de la prueba [1-5]
# $2: pagina web pedida al servidor [webPequena.html - webGrande.html -
    hola.php]

# tiempo de espera para que el servidor se descargue de trabajo entre ICs
time=10

dir='http://virtual.lab.inf.uva.es:31072/'

# uso del script
if [ "$#" -ne 2 ]; then
        echo "Uso: ./scriptAB numPrueba paginaPeticion"
        exit -1
fi

# creacion de las carpetas de salida de los datos
if [ -d ./informes/ ]; then
        echo "Ya existe la carpeta informes"
else
    echo "Carpeta informes creada"
        mkdir ./informes
        mkdir ./informes/ic1 ./informes/ic2 ./informes/ic3 ./informes/ic4
             ./informes/ic5 ./informes/ic6 ./informes/ic7 ./informes/ic8
            ./informes/ic9 ./informes/ic10 ./informes/ic11 ./informes/ic12
fi

# inicio del SADC junto con la correspondiente ruta donde se van a
    guardar los datos
ssh -p 31071 usuario@virtual.lab.inf.uva.es "sar -n DEV -r -u 5 > /home/
    usuario/Escritorio/sar/$2/carga$1" &

# inicio de las diferentes intensidades de carga
echo "IC1"
ab -k -c 50 -n 1000 -g ./informes/ic1/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic1/datos$2$1
echo "fin"
sleep $time

echo "IC2"
ab -k -c 100 -n 1000 -g ./informes/ic2/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic2/datos$2$1
sleep $time

echo "IC3"
```

```
ab -k -c 100 -n 5000 -g ./informes/ic3/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic3/datos$2$1
sleep $time

echo "IC4"
ab -k -c 250 -n 5000 -g ./informes/ic4/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic4/datos$2$1
sleep $time

echo "IC5"
ab -k -c 100 -n 10000 -g ./informes/ic5/grafico$2$1.tsvs -r -s $timeOut
    $dir$2 > ./informes/ic5/datos$2$1
sleep $time

echo "IC6"
ab -k -c 250 -n 10000 -g ./informes/ic6/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic6/datos$2$1
sleep $time

echo "IC7"
ab -k -c 100 -n 50000 -g ./informes/ic7/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic7/datos$2$1
sleep $time

echo "IC8"
ab -k -c 250 -n 50000 -g ./informes/ic8/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic8/datos$2$1
sleep $time

echo "IC9"
ab -k -c 500 -n 50000 -g ./informes/ic9/graficos$2$1.tsv -r -s $timeOut
    $dir$2 > ./informes/ic9/datos$2$1
sleep $time

echo "IC10"
ab -k -c 100 -n 100000 -g ./informes/ic10/graficos$2$1.tsv -r -s $timeOut
     $dir$2 > ./informes/ic10/datos$2$1
sleep $time

echo "IC11"
ab -k -c 250 -n 100000 -g ./informes/ic11/graficos$2$1.tsv -r -s $timeOut
     $dir$2 > ./informes/ic11/datos$2$1
sleep $time

echo "IC12"
ab -k -c 500 -n 100000 -g ./informes/ic12/graficos$2$1.tsv -r -s $timeOut
     $dir$2 > ./informes/ic12/datos$2$1
sleep $time

# ejecucion del comando para parar el monitor SADC
ssh -p 31071 usuario@virtual.lab.inf.uva.es "pkill -9 sar"

echo "Fin del Script - Mirar datos recogidos en ./informes/ic[1-12]/[web]
    "
```

ANEXO II.

Programa Java que se encarga de procesar todos los datos y mostrar los resultados.

```
import java.util.ArrayList;
import java.lang.Math;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class dataRead {

  static ArrayList<ArrayList> data = new ArrayList<ArrayList>();
  static ArrayList<Float> tTotal = new ArrayList<Float>();
  static ArrayList<Float> fallos = new ArrayList<Float>();
  static ArrayList<Float> petPorSeg = new ArrayList<Float>();
  static ArrayList<Float> tPorPet = new ArrayList<Float>();


 public static void main(String[] args) {

/*
Args:
1: Filename
2: From test # (Included)
3: To test # (Included)
*/

  String filename;

//Loads the data
  for(int i = Integer.parseInt(args[1]); i <= Integer.parseInt(args[2]);
      i++)
  {
    filename = args[0] + i + ".osda";
    readFile(filename);
  }

/*
//Debug
  System.out.println(tTotal);
  System.out.println(fallos);
  System.out.println(petPorSeg);
  System.out.println(tPorPet);
*/

//Means
  float tTotalMean = getMean(tTotal);
  float fallosMean = getMean(fallos);
  float petPorSegMean = getMean(petPorSeg);
  float tPorPetMean = getMean(tPorPet);

//Standard Deviations
  float tTotalSD = getSD(tTotal, tTotalMean);
  float fallosSD = getSD(fallos, fallosMean);
  float petPorSegSD = getSD(petPorSeg, petPorSegMean);
  float tPorPetSD = getSD(tPorPet, tPorPetMean);

//Confidence Interval (Distance from mean)
  float tTotalCI = 1.96f * (tTotalSD / (float) Math.sqrt(tTotal.size()));
  float fallosCI = 1.96f * (fallosSD / (float) Math.sqrt(fallos.size()));
  float petPorSegCI = 1.96f * (petPorSegSD / (float) Math.sqrt(petPorSeg.
      size()));
```

```
  float tPorPetCI = 1.96f * (tPorPetSD / (float) Math.sqrt(tPorPet.size()
      ));

//Removing the outer values
  for (int i = tTotal.size() - 1; i == 0; i--)
  {
    if (tTotal.get(i) < tTotalMean - tTotalCI || tTotal.get(i) >
        tTotalMean + tTotalCI)
    {
      tTotal.remove(i);
    }
  }
  for (int i = fallos.size() - 1; i == 0; i--)
  {
    if (fallos.get(i) < fallosMean - fallosCI || fallos.get(i) >
        fallosMean + fallosCI)
    {
      fallos.remove(i);
    }
  }
  for (int i = petPorSeg.size() - 1; i == 0; i--)
  {
    if (petPorSeg.get(i) < petPorSegMean - petPorSegCI || petPorSeg.get(i
        ) > petPorSegMean + petPorSegCI)
    {
      petPorSeg.remove(i);
    }
  }
  for (int i = tPorPet.size() - 1; i == 0; i--)
  {
    if (tPorPet.get(i) < tPorPetMean - tPorPetCI || tPorPet.get(i) >
        tPorPetMean + tPorPetCI)
    {
      tPorPet.remove(i);
    }
  }

//Means after CI
  float tTotalMean2 = getMean(tTotal);
  float fallosMean2 = getMean(fallos);
  float petPorSegMean2 = getMean(petPorSeg);
  float tPorPetMean2 = getMean(tPorPet);

//Prints out the mean values after the CI
  System.out.println(tTotalMean2);
  System.out.println(fallosMean2);
  System.out.println(petPorSegMean2);
  System.out.println(tPorPetMean2);

 }

//Returns the mean of a Float array
 public static float getMean (ArrayList<Float> array)
 {
  float mean = 0;
  for (int i = 0; i < array.size(); i++)
  {
    mean += array.get(i).floatValue();
  }
  mean = mean / array.size();
  return mean;
```

```
 }

//Returns the standard deviation of a Float array
 public static float getSD (ArrayList<Float> array, float mean)
 {
  float sum = 0;
  for (int i = 0; i < array.size(); i++)
  {
    sum += Math.pow(array.get(i).floatValue() - mean, 2);
  }
  sum = sum / array.size();
  sum = (float) Math.sqrt(sum);
  return sum;
 }


//Reads the file and loads the data in the class variables
 public static void readFile (String filename) {
  BufferedReader br = null;
  FileReader fr = null;

  try {
    fr = new FileReader(filename);
    br = new BufferedReader(fr);
    String sCurrentLine;

    if ((sCurrentLine = br.readLine()) != null) {
      tTotal.add(Float.valueOf(sCurrentLine));
    }
    if ((sCurrentLine = br.readLine()) != null) {
      fallos.add(Float.valueOf(sCurrentLine));
    }
    if ((sCurrentLine = br.readLine()) != null) {
      petPorSeg.add(Float.valueOf(sCurrentLine));
    }
    if ((sCurrentLine = br.readLine()) != null) {
      tPorPet.add(Float.valueOf(sCurrentLine));
    }

  } catch (IOException e) {
    e.printStackTrace();
  } finally {
    try {

      if (br != null)
      br.close();

      if (fr != null)
      fr.close();

    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
 }

}
```

ANEXO III.

Programa que se encarga de recoger todos los datos para poder pasarlos a una tabla y de ahí sacar las gráficas:

```
#!/bin/bash

#Args:
#$1: Filename

if [ $# -ne 1 ]; then
    echo "Args usage:"
    echo "  1: filename"
    exit 1
fi

for (( i=12; i<=12; i++))
        do
        ./procesar informes/ic$i/$1 1 1
        echo ""
        done
```