UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

INGENIERÍA DE TELECOMUNICACIÓN

IMPLEMENTACIÓN Y VALIDACIÓN DE UN GENERADOR DE TRÁFICO WEB

PROYECTO FIN DE CARRERA

Abril – 2008

AUTOR: Carlos Quesada Granja

DIRECTORES: Katja Gilly de la Sierra-Llamazares
Salvador Alcaraz Carrasco

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

Implementación y validación de un generador de tráfico Web

Proyectante: Carlos Quesada Granja

Director/es: Katja Gilly de la Sierra-Llamazares, Salvador Alcaraz Carrasco

VºBº director/es del proyecto:

Fdo.: Fdo.:

Lugar y fecha:

CALIFICACIÓN NUMÉRICA ☐ MATRÍCULA DE HONOR ☐

Méritos justificativos en el caso de conceder Matrícula de Honor:

Conforme presidente:	Conforme secretario:	Conforme vocal:
Fdo.:	Fdo.:	Fdo.:

Lugar y fecha:

A mis padres.
A mis hermanos Alejandro e Irene.
A Pili, Rita y sus compañeros.

A mis directores de proyecto,
en especial a Katja, por su ayuda.

Y a los que siempre me felicitan
el día de mi cumpleaños.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

La Web es uno de los servicios más importantes que ofrece Internet. De acuerdo con un estudio realizado en 2007 [1], el tráfico creado por el protocolo HTTP (empleado en cada transacción de la Web) representa entre una décima y una cuarta parte del tráfico total generado en Internet, situándose sólo por detrás de las redes de intercambio P2P. Este hecho sugiere que el número de personas que accede a contenidos web es muy elevado; por consiguiente, el rendimiento de los servidores que proporcionan dichos contenidos se convierte en un factor relevante. Si se pretende asegurar la correcta difusión de un sitio web, es fundamental verificar el rendimiento del servidor en el que se aloja para diferentes situaciones de carga, más aún si de este sitio depende un negocio o una institución.

Los benchmarks para servidores web son programas informáticos que permiten estimar el rendimiento de un servidor web para determinar su idoneidad con cargas de trabajo muy grandes. El rendimiento de un servidor se suele determinar mediante el análisis de algunos parámetros significativos, como el tiempo que necesita para servir las páginas a los usuarios, el número de peticiones que se pueden atender por unidad de tiempo o la tasa de transferencia.

1.1. ¿Qué es un benchmark?

Los benchmarks, en general, son programas que se utilizan para medir el rendimiento de un sistema informático o de alguna de sus partes. La finalidad de este tipo de estudios puede ser muy variada, como por ejemplo, la comparación de sistemas, su sintonización, la planificación de la capacidad, la comparación de compiladores en la

generación de código eficiente, el diseño de sistemas o procesadores, etc.
La medida del rendimiento de los sistemas de ordenadores implica la monitorización del sistema mientras está sujeto a una carga (*workload*) particular. Para obtener medidas significativas la carga se debe seleccionar cuidadosamente.

1.1.1. Tipos de cargas

En esta sección se describen las cargas que se usan tradicionalmente para comparar sistemas [3]. Se clasifican en cinco grupos: instrucciones de suma, mezclas de instrucciones, kernels, programas sintéticos y cargas de nivel de aplicación.

1. **Instrucciones de suma.** Inicialmente los ordenadores tenían muy pocas instrucciones, siendo la suma la de uso más frecuente. Como primera aproximación, los ordenadores con la instrucción de suma más rápida se consideraban los que mejor rendimiento ofrecían, y por tanto era la única carga usada.

2. **Mezclas de instrucciones.** Es semejante a la carga anterior, aunque en este caso se especifican varias instrucciones acompañadas de su frecuencia de uso.

3. **Kernels.** Son conjuntos de instrucciones que forman una función de alto nivel y que se utilizan como carga de prueba. Normalmente ignoran los interfaces de entrada/salida (E/S) y se limitan a medir el rendimiento del procesador; ésta es su principal desventaja, ya que no reflejan el rendimiento global del sistema. Las operaciones más típicas que realizan los kernels son la inversión de matrices, la ordenación de vectores, el cálculo de la función de Ackermann, el algoritmo de criba de Eratóstenes, etc.

4. **Programas sintéticos.** Son programas cuidadosamente diseñados para imitar —desde el punto de vista estadístico— un conjunto de programas con características similares. Los más sencillos suelen estar formados por un bucle que realiza llamadas al subsistema de E/S y que puede incorporar capacidades de medida. Normalmente se desarrollan en lenguajes de alto nivel, lo que les permite aumentar su portabilidad. Debido a su pequeño tamaño tienen el inconveniente de no ser representativos en cuanto a uso de memoria o disco. Ejemplos clásicos de benchmarks de este tipo son Whetstone y Dhrystone.

5. **Cargas de nivel de aplicación.** Se emplean para comparar el comportamiento de diferentes sistemas ante una aplicación concreta. Para ello se utilizan las funciones más representativas de esa aplicación, que suelen requerir buena parte

de los recursos del sistema, como dispositivos de E/S, acceso a redes y bases de datos. Los benchmarks para servidores web pertenecen a este tipo de cargas.

El término benchmark se utiliza casi siempre como sinónimo de carga, aunque en la bibliografía consultada [2, 3] sólo se llama benchmark a kernels, programas sintéticos y cargas de nivel de aplicación, que serán los que se analicen en la siguiente sección.

1.1.2. Ejemplos de benchmarks

Se pasa a describir algunos de los benchmarks mejor conocidos. Entre los de tipo kernel destacan el algoritmo de criba y la función de Ackermann:

- **Algoritmo de criba.** Se emplea para comparar microprocesadores, ordenadores personales, y lenguajes de alto nivel. Se basa en el algoritmo de criba de Eratóstenes y permite encontrar todos los números primos por debajo de un número dado n. El algoritmo, en su forma manual, consiste en anotar todos los enteros de 1 a n para, posteriormente, eliminar todos los múltiplos de k para $k = 2, 3, ..., n$.

- **Función de Ackermann.** Es una función recursiva que toma dos números naturales como argumentos y devuelve un número natural. Se caracteriza porque crece extremadamente rápido —por ejemplo, $A(4, 2)$ devuelve un número de casi veinte mil dígitos—. Implementada como kernel, se emplea para evaluar la eficiencia de las llamadas a funciones de los lenguajes de programación. Los parámetros que se suelen usar para comparar los sistemas son el tiempo medio de ejecución por llamada, el número de instrucciones ejecutadas por llamada y el incremento de espacio usado en memoria por llamada.

Como se comentó en la sección anterior, Whetstone y Dhrystone son ejemplos de programas sintéticos. Sus características se comentan a continuación:

- **Whetstone.** Fue el primer programa diseñado como benchmark, en 1972. Se escribió en ALGOL a partir del estudio de casi un millar de programas de carácter científico. Se caracteriza por hacer un gran uso de operaciones y datos numéricos en coma flotante; su tiempo de ejecución es corto y su código fuente sencillo. Usa pocas variables locales y, debido a su pequeño tamaño, cabe habitualmente en memoria cache. En la actualidad Whetstone aún se utiliza ampliamente, ya que proporciona una medida razonable del rendimiento de la unidad de coma flotante.

- **Dhrystone.** Creado en 1984 a partir de programas comerciales, este benchmark sintético no tiene instrucciones en coma flotante en su bucle de medida y tampoco

realiza llamadas al sistema. Gran parte del tiempo de ejecución (puede llegar al 40 %) se emplea en funciones que manejan cadenas de caracteres. Es uno de los benchmarks más usados y forma parte de la mayoría de bibliotecas diseñadas con este proposito. Existen muchas versiones de Dhrystone, algo a tener en cuenta cuando se realizan las comparaciones.

Los programas utilizados para comparar servidores web son buenos ejemplos de benchmarks de nivel de aplicación. Algunos de los más conocidos son:

- **ApacheBench.** También conocido como ab, es un programa de código abierto y multiplataforma que se incluye con el Servidor HTTP Apache [4]. Se ejecuta desde la línea de comandos y mide el rendimiento proporcionado por un servidor mediante la simulación de algunas condiciones de carga. En particular, muestra el número de peticiones por segundo que es capaz de servir una instalación.

- **httperf.** Es una herramienta para medir el rendimiento de un servidor web que ofrece varios generadores de carga [5]. Mientras se ejecuta lleva la cuenta de una serie de medidas de rendimiento que se resumen en forma de estadísticas que se muestran a su finalización. La operación más básica de httperf es generar un número fijo de peticiones y medir cuántas respuestas y con qué tasa vuelven del servidor.

- **The Grinder.** Es un entorno de trabajo en Java para el estudio de cargas que permite la ejecución de pruebas distribuidas usando varias máquinas inyectoras de tráfico [6]. Cada prueba se debe implementar mediante un guión (*script*) en el lenguaje de programación Jython. El programa realiza una gestión de las conexiones HTTP muy eficiente, y dispone de una consola gráfica que permite monitorizar y controlar los inyectores de carga. Tiene una licencia de código abierto.

- **Web Capacity Analisys Tool** y **Web Application Stress Tool.** Son dos herramientas de Microsoft para los sistemas operativos Windows que simulan cargas en aplicaciones web [7]. La primera se integra dentro de un paquete mayor y se utiliza para comprobar y planificar la capacidad de los servidores y las redes mediante simulaciones de carga. La segunda está diseñada para simular con realismo las peticiones de varios navegadores a las páginas de un sitio web.

1.2. Objetivo del proyecto

Este proyecto tiene como objetivo la implementación y posterior validación de un benchmark para servidores web, que además de generar una carga con un comportamiento preestablecido, deberá simular las características de las páginas alojadas en un servidor web y observar su rendimiento.
Así, el benchmark desarrollado estará constituido por un conjunto de programas, que deberán encontrarse instalados tanto en el equipo que ejecutará el generador de carga (el equipo cliente), como en el equipo cuyo rendimiento se desea conocer (el equipo servidor de páginas web). Obviamente, estos equipos deberán estar accesibles entre sí en una red de ordenadores. Las características que presentarán los programas desarrollados se enumeran a continuación:

1. La carga generada por el benchmark estará formada por un determinado número de peticiones HTTP dirigidas al servidor web. Cada petición solicitará la descarga de una página web concreta.

2. El programa se conectará con el servidor vía SSH (*Secure Shell*) momentos antes de hacer el benchmarking para crear el conjunto de páginas web sobre las que se realizarán las peticiones.

3. Las características de las páginas web que se crearán en el servidor y el comportamiento de las peticiones realizadas por el benchmark serán parámetros que deberá definir el usuario en función de unas distribuciones de probabilidad dadas.

4. Las páginas web creadas en el servidor podrán ser estáticas o dinámicas. Las páginas estáticas sólo contendrán código HTML, mientras que las dinámicas también añadirán PHP y harán uso del sistema de gestión de bases de datos MySQL. Todas las páginas se compondrán de objetos, que simularán los elementos HTML (marcos, tablas, listas), imágenes, archivos empotrados, etc. de los que se componen las páginas web reales.

5. Mientras se esté ejecutando el benchmark, un programa monitorizará en el servidor el porcentaje de CPU consumido cada segundo. En el cliente se medirán los tiempos de respuesta de los ficheros descargados y sus tamaños.

6. Al finalizar la ejecución, el programa creará dos ficheros con los resultados obtenidos. Uno de ellos estará en formato CSV (*Comma-Separated Values*), para facilitar el tratamiento de los datos con entornos de análisis numérico.

Para validar el benchmark se realizarán varias series de pruebas sobre tres topologías de red diferentes. Se elaborarán diversas estadísticas y gráficas con los resultados obtenidos, que se compararán con los valores teóricos previstos para comprobar que los programas funcionan correctamente.

Capítulo 2

IMPLEMENTACIÓN

Éste es un capítulo extenso en el que se documentan los métodos, algoritmos y protocolos que se han aplicado para llevar a cabo el benchmark.
En la sección 2.1 se describen las características de los tres programas que conforman el benchmark. En la sección 2.2 se detallan los protocolos de comunicación más importantes que usan los programas desarrollados. En la sección 2.3 se explica cómo se ha abordado la generación de números aleatorios para las cinco distribuciones de probabilidad que el usuario puede seleccionar. En la sección 2.4 se explica el modo en el que se crean las páginas web en el servidor, que es diferente dependiendo de si son estáticas o dinámicas. Y por último, en las secciones 2.5 a 2.8 se comenta la estructura y el código de los ficheros fuente que componen el benchmark.

2.1. Descripción de los programas

El benchmark se ha dividido en tres programas diferentes: un generador de carga, un generador de páginas web y un monitor de procesos. Los tres se han escrito en el lenguaje de programación C.
El **generador de carga** (`qb`) crea tráfico HTTP dirigido al servidor y recoge los datos obtenidos durante la ejecución para elaborar un informe final. Además, controla de forma transparente al usuario los otros dos programas, necesarios para el correcto funcionamiento del benchmark.
El **generador de páginas web** (`qgen`) se ejecuta en el equipo servidor bajo las órdenes del generador de carga. Antes de que comience el benchmarking, este generador crea un conjunto de páginas web con unas características definidas por el usuario. Son éstas

las páginas a las que irán dirigidas las peticiones del generador de carga.
El **monitor de procesos** (`qmon`), al igual que el generador de páginas web, se ejecuta en el servidor y su funcionamiento está controlado por el generador de carga. Realiza un seguimiento del consumo de la CPU del equipo por parte de los procesos relacionados con la gestión de las páginas web.

2.1.1. Tecnologías empleadas

Los programas que componen el benchmark constituyen una solución LAMP. Con ese acrónimo se conoce al conjunto de *software* libre y de código abierto empleado en aplicaciones que hacen uso de servidores dinámicos; éstos son: **L**inux, **A**pache, **M**ySQL y **P**HP.

- **Linux.** Se ha elegido este sistema operativo por su fiabilidad, su seguridad y, sobre todo, por la comodidad que ofrece en el desarrollo de aplicaciones. Todo el *software* necesario para que los programas funcionen se puede obtener con sencillez, mediante un gestor de paquetes. Además, se puede encontrar en Internet mucha documentación sobre cualquier tema relacionado.

- **Apache HTTP Server.** Es un popular servidor web que permitirá atender las peticiones HTTP de los clientes. Por defecto, las páginas web que sirve se alojan en la carpeta `/var/www`, y responde a las peticiones por el puerto 80.

- **MySQL.** Es un sistema de gestión de bases de datos que se utilizará junto con el servidor de páginas web para poder servir páginas web dinámicas. Con los archivos de instalación del benchmark se incluye la base de datos necesaria para generar fragmentos de texto de tamaño variable.

- **PHP.** Es un lenguaje de programación interpretado que se utiliza para la creación de páginas web dinámicas. Se integra en HTML y se ejecuta en un servidor web. Tiene instrucciones que permiten realizar consultas a bases de datos.

Aunque estos programas no se diseñaron específicamente para trabajar juntos, lo cierto es que su combinación se ha hecho muy popular.
Otros recursos relevantes empleados en la implementación del benchmark son:

- **HTML.** Es un lenguaje de marcado diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

- **OpenSSH.** Aplicación de código abierto que permite el acceso y la administración de máquinas remotas gracias a comunicaciones cifradas, proporcionadas por el protocolo SSH. Posibilita la comunicación del generador de carga en el equipo cliente con el generador de páginas web y el monitor de procesos en el equipo servidor.

2.1.2. Instalación y manual de uso

La instalación del benchmark se detalla en el apéndice B. Los archivos necesarios se encuentran en el DVD adjunto (el contenido de este disco se especifica en el apéndice A). El benchmark se ejecuta escribiendo el nombre del programa generador de carga desde la línea de comandos de Linux (`./qb`), acompañado de una serie de parámetros de entrada, como son: la dirección IP de la máquina que hace de servidor web, el usuario del proceso que ejecuta dicho servidor, el número de peticiones que se quieren hacer al servidor, las características de la distribución de probabilidad que determinará el tiempo transcurrido entre peticiones, el número de páginas que se tienen que crear en el servidor, su tipo, su tamaño, etc.
Los otros dos programas —el generador de páginas web y el monitor de procesos—, que se ejecutan al mismo tiempo que `qb` y de forma automática, también pueden ejecutarse por separado si se desea.
Las instrucciones de uso con los parámetros de entrada válidos para los tres programas se encuentran en el apéndice C.

2.1.3. Secuencia de ejecución del benchmark

La primera tarea que realiza el generador de carga cuando se ejecuta es comprobar que las opciones introducidas por el usuario son correctas. Si no lo fueran, el programa mostraría un mensaje de error y se terminaría.
A continuación, el generador de carga realiza una conexión SSH con el equipo servidor para que éste ejecute el generador de páginas web. En la conexión se incluyen los datos proporcionados por el usuario relacionados que el tipo de páginas que se quieren crear (estáticas o dinámicas), su tamaño, el número de objetos que las componen, etc. (el modo en el que se crean las páginas web se especifica en la sección 2.4). Si el generador de páginas web se ha ejecutado correctamente en el servidor, el cliente obtiene un fichero de texto (`qgen.txt`) con las direcciones y características de las páginas creadas. Este fichero permite que el generador de carga conozca las páginas web hacia las que debe dirigir el tráfico.

El siguiente paso previo al benchmarking, consiste en realizar una segunda conexión SSH con el servidor, esta vez para ejecutar el monitor de procesos. Este programa observa y anota el uso que hacen de la CPU los procesos del servidor Apache, y, en el caso de las páginas dinámicas, también los de MySQL.
A partir de este momento, el generador de carga comienza a enviar las peticiones HTTP al servidor. Los instantes en los que se envía cada petición vienen determinados por la distribución de probabilidad escogida por el usuario (las distribuciones de probabilidad disponibles y sus características se comentan en la sección 2.3). Las páginas web se descargan completamente, es decir, el programa simula ser un navegador y realiza peticiones por cada uno de los objetos contenidos en una página. Durante todo este proceso, el programa lleva la cuenta del tiempo de descarga de las páginas y de cada uno de los objetos que componen dichas páginas, así como del tamaño de las páginas y los objetos descargados.
Cuando se han enviado todas las peticiones, el programa realiza una tercera y última conexión SSH con el servidor para detener el monitor de procesos. Por último, crea un par de archivos (sus nombres por defecto son `result.txt` y `result.csv`) con los datos recogidos durante las descargas para su posterior análisis.

2.2. Aplicación de los protocolos de comunicación

El tráfico web depende de una serie de protocolos de comunicación. Un protocolo define la sintaxis y la semántica de los mensajes intercambiados entre emisores y receptores. Los protocolos de comunicación se suelen desarrollar por *capas*, manejando cada una de ellas un aspecto específico de la comunicación. El modelo de referencia de Internet se estructura en cuatro capas principales [8]:

- **Capa de enlace:** Trata los detalles de hardware del interfaz con el medio de comunicación físico, como Ethernet, el Modo de Transferencia Asíncrona (ATM) o Red Óptica Síncrona (SONET).
- **Capa de red:** Trata la entrega de *paquetes* de datos a través de una red. Los protocolos de capa de red se implementan en los *routers* y *hosts* finales.
- **Capa de transporte:** Coordina la comunicación entre ordenadores en nombre de la capa de aplicación. En la práctica, los protocolos de la capa de transporte se implementan en el sistema operativo del *host* final.
- **Capa de aplicación:** Trata los detalles de aplicaciones específicas. En la práctica, un protocolo de capa de aplicación se implementa como parte del software de

aplicación, como un navegador web o un servidor web.

Esta división de funcionalidad permite que cada protocolo esté enfocado en la realización de una única tarea con interfaces bien definidos a los protocolos de capas adyacentes. La estandarización de los protocolos posibilita la interoperabilidad entre componentes desarrollados por diferentes vendedores.
En las siguientes subsecciones se presenta una visión general de los cuatro protocolos de comunicación más importantes que se han tenido que gestionar en el desarrollo del benchmark. Éstos son: el Protocolo de Internet (IP), el Protocolo de Control de Transmisión (TCP), el Protocolo de Transferencia de Hipertexto (HTTP) y Secure Shell (SSH). De todos ellos, sólo los dos primeros se encuentran implementados en bibliotecas de C. A continuación, se comentan estos protocolos.

2.2.1. TCP e IP

El Protocolo de Control de Transmisión (TCP) y el Protocolo de Internet (IP) son dos de los protocolos más utilizados del modelo de referencia de Internet:

- **IP** es el protocolo de capa de red usado en Internet. Coordina la entrega de paquetes individuales de un equipo a otro a partir de la dirección del destinatario (dirección IP). Ofrece un servicio no orientado a conexión y no confirmado; es decir, la información se puede perder, duplicar o desordenar.

- **TCP** es el principal protocolo de capa de transporte de Internet. Coordina la transmisión de los paquetes IP entre dos equipos para que proporcionen una comunicación confiable y bidireccional mediante el uso de números de puerto. Este protocolo es fundamental en sistemas de comunicación, ya que modela el comportamiento de un canal ideal.

TCP e IP se aplican en C mediante instrucciones de gestión de ***sockets***. Un *socket* es un mecanismo informático que proporciona un canal de comunicación entre dos procesos, incluso cuando éstos se encuentran en diferentes ordenadores. Los procesos implicados referencian a un *socket* mediante un **descriptor**, del mismo tipo que los usados para referenciar ficheros. La comunicación de procesos a través de *sockets* se basa en una arquitectura cliente-servidor, y queda definida por un protocolo (normalmente TCP o UDP), dos direcciones IP y dos números de puerto (los del equipo local y el remoto).
Un poco más adelante, en la sección 2.2.4, se explica detalladamente cómo gestiona los *sockets* el benchmark.

2.2.2. HTTP

HTTP es un protocolo de capa de aplicación que permite la transferencia de recursos a través de la Web. Sigue un esquema petición-respuesta y sin estado: el cliente envía una petición y el servidor la responde, pero cada transacción se trata de forma independiente.

En el protocolo HTTP, las peticiones realizadas por el cliente se descomponen en tres partes. La primera de ellas está formada por un método HTTP, seguido de una dirección de documento o URI (*Uniform Resource Identifier*) y un número de versión HTTP, de forma que se tiene una línea de texto con el formato "`Comando URI Protocolo`". Por ejemplo:

```
GET /index.html HTTP/1.1
```

El único método implementado en este benchmark es `GET`, que permite solicitar al servidor un documento existente en su espacio de direcciones.

Tras la petición, el cliente puede enviar información adicional de cabeceras con las que se da al servidor más información, como el tipo de *software* que ejecuta el cliente, el tipo de contenido que entiende el cliente, etc. Esta información puede ser utilizada por el servidor para generar la respuesta apropiada. Las cabeceras se envían una por línea, donde cada una tiene el formato "`Clave:Valor`". Por ejemplo:

```
If-Modified-Since: Sat, 24 Nov 2007 19:08:01 GMT
Referer: http://www.w3.org/hypertext/DataSources/Overview.html
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

Después de las cabeceras, el cliente envía una línea en blanco para indicar el final de la sección. Por último, de forma opcional, se pueden enviar datos adicionales si el método de HTTP utilizado lo requiere.

El protocolo ha pasado por muchas versiones. En la actual, HTTP/1.1, están activadas por defecto las conexiones persistentes (véase el esquema de la figura 2.1) y funcionan bien con los *proxies*. También soporta peticiones en paralelo (*pipelining*), que permiten el envío de varias peticiones simultáneamente.

Para el diseño del benchmark se ha empleado un esquema de conexiones múltiples, ya que permite cronometrar una a una las peticiones. Para evitar el esquema de conexión persistente que viene por defecto en la versión HTTP/1.1, el generador de carga usa por simplicidad la versión anterior, HTTP/1.0.

Las peticiones concretas que realizará el benchmark serán similares a la siguiente:

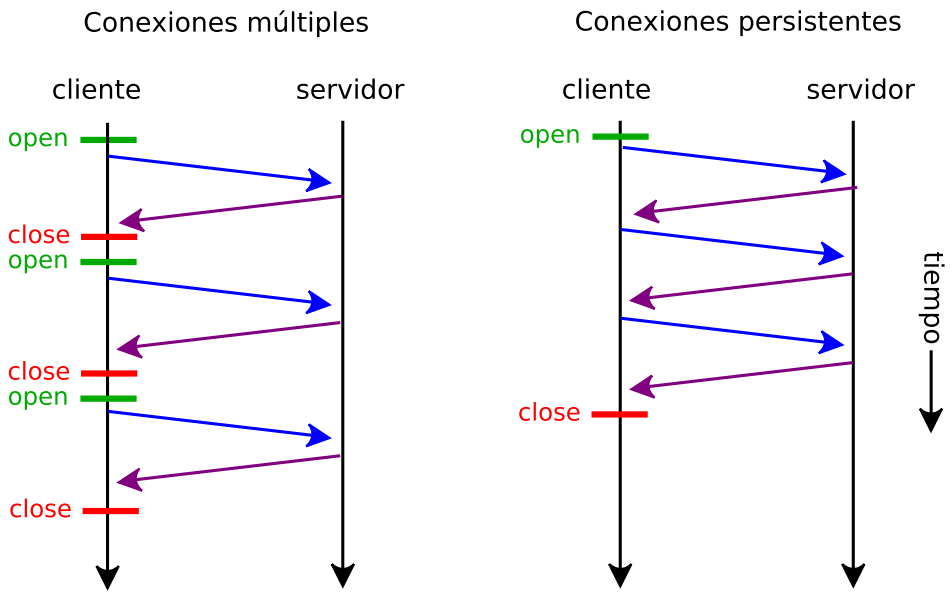

Figura 2.1: Comparación entre conexiones múltiples y persistentes de HTTP.

```
GET /obj0004.html HTTP/1.0
Host: 192.168.1.133:80
Connection: close
```

En este caso, el cliente está solicitando al servidor web de la dirección `192.168.1.133` la descarga del archivo `obj0004.html`. Con la cabecera `Connection:close` se está indicando que la conexión se debe cerrar tras completar la respuesta. De otra forma la conexión no se cerraría hasta agotar un tiempo de espera.

2.2.3. SSH

Secure Shell es un protocolo de capa de aplicación que permite el intercambio de datos sobre un canal seguro entre dos ordenadores. SSH usa criptografía de clave pública (o asimétrica) para autenticar al equipo remoto, lo que proporciona confidencialidad e integridad a los datos.

El benchmark usará SSH para que el generador de carga se pueda registrar de forma remota en el servidor cada vez que sea necesario y, desde allí, ejecutar el generador de páginas web y el monitor de procesos. Para aplicar el protocolo, se ha optado por ejecutar el programa de código abierto OpenSSH como un comando de consola.

Es conveniente configurar OpenSSH como se explica en el apéndice B.4 para que el benchmark no pregunte la contraseña del servidor cada vez que se intenta acceder a él.

2.2.4. Gestión de sockets

En este benchmark, el generador de carga se comporta como un cliente web que establece un *socket* con una aplicación remota: el servidor de páginas web de Apache. El generador de carga inicia la creación de un *socket* mediante la instrucción `socket`. A continuación, la aplicación llama a `connect` para asociar el *socket* con la dirección IP y el número de puerto del servidor Apache. Como parte de la ejecución de la función `connect`, el sistema operativo también selecciona un número de puerto local no utilizado (entre el 1024 y el 65535) para el generador de carga.

En este punto, el sistema operativo que ejecuta el generador de carga conoce las dos direcciones IP y los dos números de puerto que identifican unívocamente la conexión bidireccional entre las dos aplicaciones. Entonces, el sistema operativo inicia el establecimiento de la conexión TCP al servidor Apache. Tras el establecimiento de la conexión, la llamada a `connect` termina y el programa puede comenzar a leer y escribir datos en el *socket*.

Una vez que la conexión se ha establecido, cualquiera de las aplicaciones puede leer o escribir datos. De hecho, ambas aplicaciones pueden leer y escribir a la misma vez porque el socket proporciona un canal de comunicación bidireccional. Con las páginas web, el cliente es quien inicia la comunicación.

El cliente (el generador de carga) escribe la petición HTTP en el *socket*. El servidor (Apache) espera a que el dato llegue a su *socket*. Entonces, una vez que el dato ha llegado, el servidor lee la petición HTTP del *socket*. Tras procesar la petición, el servidor escribe la respuesta HTTP en el *socket*. Mientras tanto, el cliente espera a que el dato llegue al *socket*. Entonces el cliente lee la respuesta del *socket*. Este patrón es el típico en aplicaciones cliente-servidor.

El sistema operativo maneja los detalles del establecimiento de la conexión lógica entre las dos aplicaciones y de la coordinación de la transferencia de paquetes. Con el generador de carga, el sistema operativo ejecuta las funciones `socket` y `connect`. Si el servidor remoto no responde, el sistema operativo informa al generador que la petición para crear un *socket* ha fallado. Con Apache, el sistema operativo ejecuta las funciones `socket`, `bind`, `listen` y `accept`.

Cuando se crea un *socket*, cada nodo asigna memoria para transmitir y recibir paquetes. Cuando la aplicación escribe en el *socket*, el sistema operativo dirige el dato a la dirección IP y puerto remotos. Asimismo, cuando los paquetes llegan, el sistema operativo dirige los datos al *socket* apropiado basándose en la numeración de los puertos. La aplicación puede entonces leer los datos desde el *socket*. Por debajo de las dos aplicaciones comunicantes, el sistema operativo coordina el envío y recepción de paquetes

para crear la abstracción del flujo de bits ordenado y confiable.

2.3. Generación de variables aleatorias

El tiempo transcurrido entre peticiones, el número de objetos que componen una página y el tamaño de cada uno de estos objetos se modelan como variables aleatorias de distribuciones de probabilidad cuyos principales parámetros estadísticos deben ser definidos por el usuario. Se han implementado cinco distribuciones de probabilidad: uniforme, exponencial, hiperexponencial, de Pareto y lognormal.
En la siguiente sección se comentan las propiedades deseables de un generador de números aleatorios para, a continuación, analizar las distribuciones de probabilidad que se han elegido y plantear los métodos de simulación que se han aplicado en los programas.

2.3.1. Generación de números pseudoaleatorios

Casi todos los métodos de simulación necesitan de un generador de números aleatorios con distribución uniforme en el intervalo $(0, 1)$ —de ahora en adelante $U(0, 1)$—. Hasta el desarrollo de los ordenadores, los números aleatorios se obtenían mediante procedimientos experimentales (loterías, ruletas) y se almacenaban en tablas. En la actualidad, estos números se generan por ordenador y se denominan **pseudoaleatorios** ya que, en realidad, todos los números generados en una sucesión pueden predecirse a partir del primero, llamado **semilla**. Todo generador de números pseudoaleatorios se debe comportar como una muestra de datos independientes de una distribución $U(0, 1)$.
Como se estudia en [9], los requisitos deseables que todo generador de números pseudoaleatorios debe satisfacer se pueden resumir en cinco puntos:

1. **Superación de las pruebas de aleatoriedad e independencia más habituales.** La aleatoriedad hace referencia a la ausencia de sesgo o correlación. La independencia indica que la ocurrencia de un suceso no influye en la probabilidad de ocurrencia del siguiente. Para comprobar estas dos propiedades en un generador se suele recurrir a pruebas de tipo chi-cuadrado (χ^2), que miden la discrepancia entre una distribución teórica y la observada.

2. **Reproducibilidad de la sucesión generada a partir de la semilla.** Permite que una misma secuencia se pueda ejecutar en diferentes ocasiones. La reproducibilidad va ligada a la pseudoaleatoriedad de los generadores: si una secuencia

es reproducible entonces no puede ser perfectamente aleatoria, ya que a partir de los valores anteriores se pueden averiguar los siguientes.

3. **Disposición de una longitud de ciclo tan grande como se desee.** Un generador de números pseudoaleatorios es una máquina de estados finitos con, como máximo, 2^p estados diferentes, donde p es el número de bits que representa el estado. La secuencia se repetirá tras la generación de los diferentes 2^p números. El menor número de pasos tras los que el generador comenzará a repetirse se conoce con el nombre de periodo o longitud de ciclo, y conviene que sea lo mayor posible.

4. **Generación de valores a gran velocidad.** Es deseable que los números se generen lo más rápido posible. Aunque en algunas aplicaciones la generación de los números pseudoaleatorios es el factor limitante, muchos generadores sólo necesitan unos cuantos ciclos de reloj. La velocidad no suele ser uno de los problemas más importantes.

5. **Bajo consumo de memoria.**

El benchmark obtiene los números pseudoaleatorios con distribución $U(0,1)$ mediante la función `drand48()`, que se encuentra implementada en la biblioteca estándar de C `stdlib.h`. Esta función pertenece a una familia de funciones generadoras de números pseudoaleatorios que emplea un algoritmo congruente y lineal que trabaja con números enteros de 48 bits (de ahí su nombre). La ecuación particular que emplea es

$$r(n+1) = (ar(n) + c) \bmod m \tag{2.1}$$

donde los valores por defecto son $a = 68\,164\,576\,877$ y $c = 22$; el módulo es siempre fijo y vale $m = 2^{48}$. Con $r(n)$ se denota la semilla del generador de números pseudoaleatorios. Por cada llamada a la función `drand48()` se procesa una iteración del algoritmo.
La función `drand48()` no recibe ningún parámetro de entrada y devuelve valores de tipo `double` (64 bits en coma flotante: 1 bit de signo, 11 de exponente y 52 de mantisa). Los 48 bits de $r(n+1)$ se cargan en los 48 bits más significativos de la mantisa del valor devuelto (los 4 bits restantes se ponen a cero). El exponente se dispone de tal forma que los todos los valores producidos recaen en el intervalo semiabierto $[0, 1)$.
La función `srand48()` permite modificar la semilla. Tiene como parámetro de entrada una variable de tipo `long int` (32 bits), que se emplea para inicializar el *buffer* interno $r(n)$ de `drand48()`, de tal manera que los 32 bits del valor de la semilla se copian en los 32 bits más significativos de $r(n)$, mientras que los 16 bits menos significativos

se establecen arbitrariamente a 13 070. Además, las constantes a y c del algoritmo se reinician a sus valores por defecto.
La semilla que el benchmark establece por defecto es el valor devuelto por la función `time`, implementada en la biblioteca estándar de C `time.h`, que representa el número de segundos transcurridos desde la medianoche del 1 de enero de 1970. De esta forma se consigue que el generador tenga una semilla diferente cada segundo.

2.3.2. Métodos de simulación de distribuciones continuas

El método más común para simular distribuciones continuas es el **método de inversión** (también conocido en algunos ámbitos como método de Montecarlo). Se apoya en el siguiente teorema:

> Sea X una variable aleatoria con función de distribución acumulada F, continua e invertible. Si $U = U(0,1)$, entonces la variable aleatoria $F^{-1}(U)$ tiene función de distribución F (la misma distribución que la de X).

A partir de este teorema, cuya demostración se puede encontrar en [9], se puede aplicar el siguiente algoritmo para simular cualquier variable continua con función de distribución acumulada F invertible:

1. Generar $U \sim U(0,1)$.
2. Devolver $X = F^{-1}(U)$.

Por tanto, una condición mínima para la aplicación de este método es conocer la forma explícita de F^{-1}. De las cinco distribuciones que se han simulado, esto ocurre para la uniforme, la exponencial y la de Pareto.
Para simular la distribución lognormal se ha recurrido a una adaptación de la **transformada de Box-Muller**, un método que genera pares de números aleatorios e independientes con distribución normal (de media cero y varianza unidad) a partir de una fuente de números con distribución $U = U(0,1)$. El algoritmo se explica en la sección 2.3.7.

2.3.3. Distribución uniforme

Ésta es una de las distribuciones más sencillas de calcular. Se suele emplear para simular una variable aleatoria con unos límites conocidos de la que no se tiene más información disponible. Por ejemplo, la distancia entre la fuente y el destinatario de un mensaje en una red o el tiempo de búsqueda en un disco se suelen modelar con esta distribución.

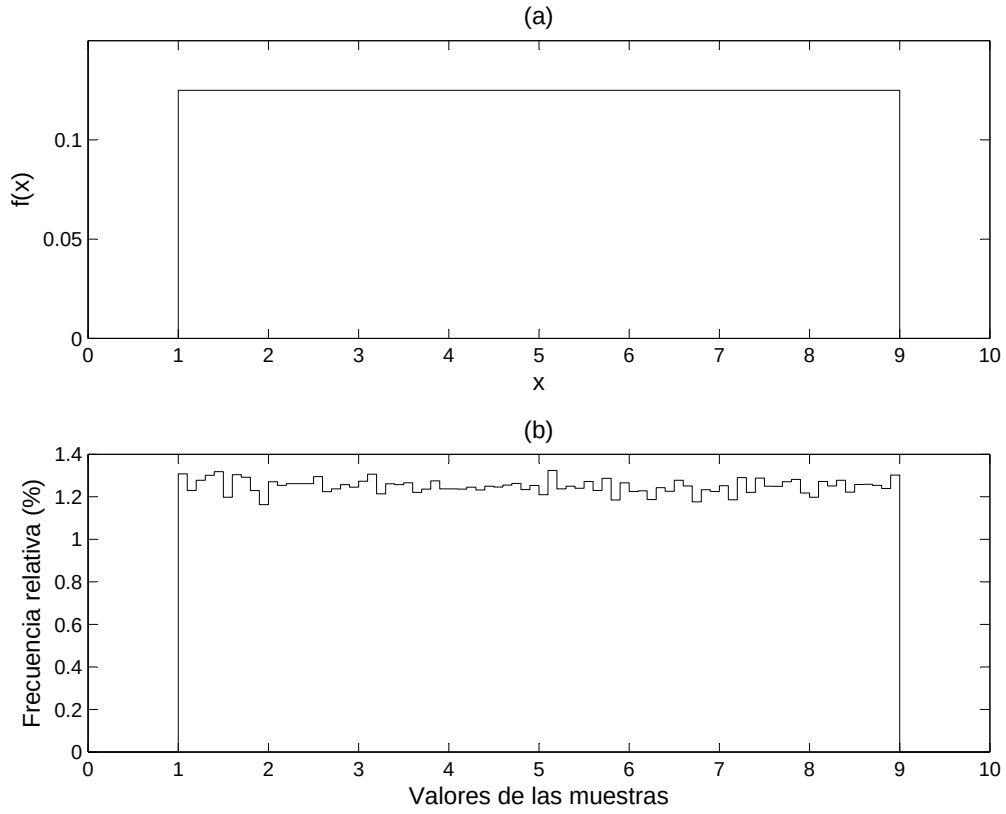

Figura 2.2: Función de densidad de la distribución uniforme.

Función de densidad de probabilidad, media y varianza

La función de densidad de probabilidad de la distribución uniforme viene dada por

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{para } a \leq x \leq b, \\ 0 & \text{para cualquier otro punto} \end{cases} \tag{2.2}$$

siendo a y b los límites inferior y superior respectivamente, con $a, b \in (-\infty, \infty)$ y $b > a$. En la implementación del benchmark estos límites se restringen a $a \geq 0$ y $b \geq 0$, ya que los valores negativos no son aceptables. Se permite también que $a = b$; en este caso la variable devuelve siempre el mismo valor. En la figura 2.2 (a) se puede ver la función de densidad de probabilidad de una distribución uniforme particularizada para $a = 1$ y $b = 9$.

La media y la varianza de una distribución uniforme se calculan como

$$\mu = \frac{a+b}{2} \qquad \sigma^2 = \frac{(b-a)^2}{12} \tag{2.3}$$

Obtención del algoritmo de simulación

La función de distribución acumulada de una variable aleatoria real X se puede definir en términos de la función de densidad de probabilidad f de la siguiente manera

$$F(x) = \int_{-\infty}^{x} f(t)\,dt \tag{2.4}$$

Para la distribución uniforme se obtiene

$$F(x) = \begin{cases} 0 & \text{para } x < a, \\ \frac{x-a}{b-a} & \text{para } a \leq x < b, \\ 1 & \text{para } x \geq b \end{cases} \tag{2.5}$$

que es continua e invertible en el intervalo $[a, b)$. Aplicando el método de inversión visto en la sección 2.3.2 se puede observar que

$$\begin{aligned} x = F^{-1}(u) \Leftrightarrow F(x) = u \Leftrightarrow \frac{x-a}{b-a} = u \\ \Leftrightarrow x - a = u\,(b-a) \Leftrightarrow x = u\,(b-a) + a \end{aligned} \tag{2.6}$$

El algoritmo que se obtiene del resultado anterior queda como sigue

1. Generar $U \sim U(0, 1)$.
2. Devolver $X = U(b - a) + a$.

Para ahorrar cálculos cuando se llama muchas veces al generador, es conveniente definir inicialmente la variable $L = b - a$, que evitará la operación de resta en llamadas sucesivas. De esta forma, la versión optimizada del algoritmo resulta

0. Hacer $L = b - a$.
1. Generar $U \sim U(0, 1)$.
2. Devolver $X = U \cdot L + a$.
3. Repetir los pasos 1–2 tantas veces como se precise.

En la figura 2.2 (b) se muestra un histograma de frecuencia relativa, de cien clases iguales, elaborado con cien mil muestras obtenidas a partir del algoritmo anterior, con $a = 1$ y $b = 9$.

2.3.4. Distribución exponencial

La distribución de probabilidad exponencial se suele usar para modelar el tiempo entre eventos independientes que suceden con un promedio constante. Tiene un uso muy extendido en teoría de colas, ya que permite modelar los tiempos entre llegadas de clientes a un servicio.

Función de densidad de probabilidad, media y varianza

La función de densidad de la distribución exponencial es un caso particular de la función de densidad gamma [19, 3], y tiene la siguiente forma

$$f(x) = \begin{cases} \frac{e^{-x/\beta}}{\beta} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{2.7}$$

donde β se conoce como *parámetro de escala* de la distribución, y debe cumplir que $\beta > 0$.
La mayoría de los autores consultados emplean una parametrización alternativa, que consiste en realizar la sustitución $\lambda = \beta^{-1}$. La función de densidad de probabilidad queda como sigue

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{2.8}$$

La variable λ recibe entonces el nombre de *parámetro de tasa* y también debe cumplir que $\lambda > 0$. Esta expresión es más compacta y su uso está más extendido que la ecuación 2.7. En la figura 2.3 (a) se puede ver la función de densidad de probabilidad de una distribución exponencial particularizada para $\lambda = \beta = 1$.
La media y la varianza de la distribución exponencial son

$$\mu = \beta = 1/\lambda \qquad \sigma^2 = \beta^2 = 1/\lambda^2 \tag{2.9}$$

Obtención del algoritmo de simulación

Aplicando la ecuación 2.4 en 2.8 se obtiene la función de distribución acumulada, que es

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{2.10}$$

Puesto que la ecuación anterior es continua e invertible en el intervalo $[0, \infty)$, se puede

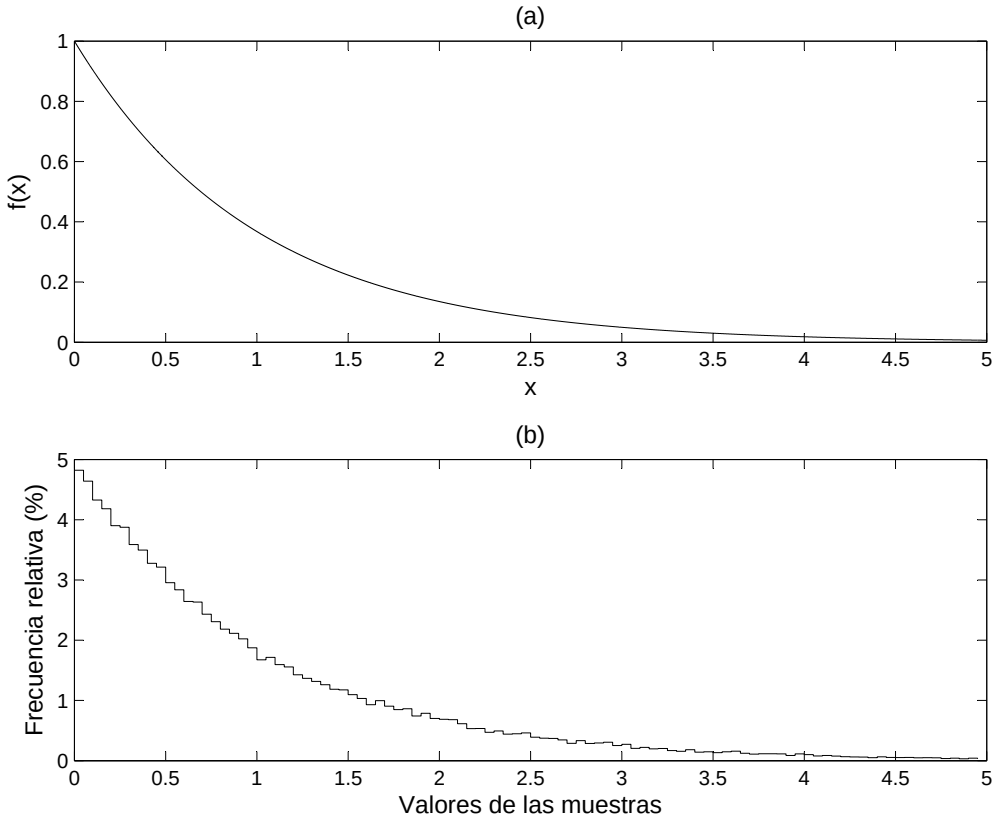

Figura 2.3: Función de densidad de la distribución exponencial.

aplicar el método de inversión

$$\begin{aligned} x = F^{-1}(u) \Leftrightarrow F(x) = u \Leftrightarrow 1 - e^{-\lambda x} = u \\ \Leftrightarrow 1 - u = e^{-\lambda x} \Leftrightarrow \ln(1-u) = \ln e^{-\lambda x} \\ \Leftrightarrow \ln(1-u) = -\lambda x \Leftrightarrow x = -\frac{1}{\lambda}\ln(1-u) \end{aligned} \tag{2.11}$$

El algoritmo que se deduce del resultado obtenido es el siguiente:

0. Hacer $L = -1/\lambda$ (de forma alternativa, $L = -\mu$).
1. Generar $U \sim U(0,1)$.
2. Devolver $X = L \cdot \ln U$.
3. Repetir los pasos 1–2 tantas veces como se precise.

Nótese que el punto 2 del algoritmo anterior se ha escrito como $X = L \cdot \ln U$ y no como $X = L \cdot \ln(1-U)$, que es lo que debería ser en base al desarrollo descrito en (2.11). Esto se ha hecho para evitar una resta en cada repetición del algoritmo, lo que reduce

el tiempo de cálculo. El cambio no afecta a la validez del método, pues si $U = U(0,1)$ entonces $1-U = U(0,1)$ y, por tanto, $g(U) = g(1-U)$ para cualquier transformación g.

En la figura 2.3 (b) se muestra un histograma de frecuencia relativa, de cien clases iguales, elaborado con cien mil muestras obtenidas a partir del algoritmo anterior, con $\lambda = \mu = 1$.

2.3.5. Distribución hiperexponencial

Función de densidad de probabilidad, media y varianza

La función de densidad de probabilidad de una variable aleatoria X con distribución hiperexponencial se expresa como

$$f_X(x) = \sum_{i=1}^{n} p_i \cdot f_{Y_i}(y) \tag{2.12}$$

donde Y_i es una variable aleatoria con distribución exponencial, con un parámetro de tasa λ_i, y p_i es la probabilidad de que X tome la forma de la distribución exponencial con tasa λ_i.

Recibe el nombre de distribución hiperexponencial porque su *coeficiente de variación* (una medida de la dispersión de una distribución de probabilidad, definida por $c_v = \sigma/\mu$, con σ la desviación estándar y μ la media) es mayor que el de la distribución exponencial, cuyo coeficiente de variación es 1, y que el de la distribución hipoexponencial, con un coeficiente de variación menor que uno.

Un ejemplo de variable aleatoria hiperexponencial se puede encontrar en el contexto de la telefonía. Por ejemplo, el tiempo uso de una línea telefónica en la que pueden concurrir un teléfono y un módem (pero no de forma simultánea) se puede modelar como una distribución hiperexponencial, ya que hay una probabilidad p de hablar por el teléfono con tasa λ_1 y una probabilidad $q = 1 - p$ de usar la conexión a Internet con tasa λ_2.

El benchmark se ha diseñado para que $n = 2$ (como en el ejemplo anterior). Esto permite que la función de distribución de probabilidad se pueda rescribir de una forma más sencilla:

$$f_X(x) = p \cdot f_{Y_1}(y) + (1-p) \cdot f_{Y_2}(y) \tag{2.13}$$

El valor medio y la varianza de esta distribución se obtienen con las expresiones

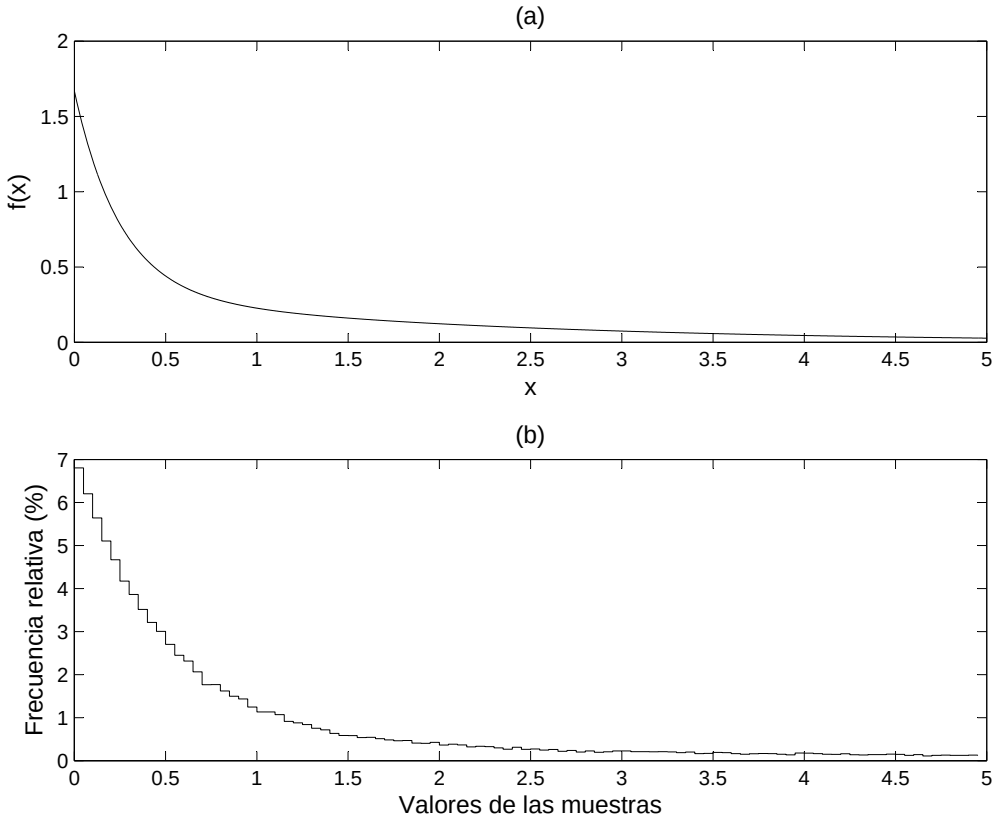

Figura 2.4: Función de densidad de la distribución hiperexponencial.

$$\mu = \sum_{i=1}^{n} \frac{p_i}{\lambda_i} \qquad \sigma^2 = 2 \cdot \sum_{i=1}^{n} \frac{p_i}{\lambda_i^2} - \left(\sum_{i=1}^{n} \frac{p_i}{\lambda_i} \right)^2 \tag{2.14}$$

que para el caso particular de $n = 2$ dan como resultado

$$\mu = \frac{p}{\lambda_1} + \frac{1-p}{\lambda_2} \qquad \sigma^2 = 2 \left(\frac{p}{\lambda_1^2} + \frac{1-p}{\lambda_2^2} \right) - \left(\frac{p}{\lambda_1} + \frac{1-p}{\lambda_2} \right)^2 \tag{2.15}$$

Obtención del algoritmo de simulación

El algoritmo para obtener números aleatorios con distribución hiperexponencial (para $n = 2$) es similar al que ya se ha visto en la distribución exponencial, salvo en lo referente a las probabilidades p_i:

0. Hacer $L_1 = -1/\lambda_1$ y $L_2 = -1/\lambda_2$.

1. Generar $U_1, U_2 \sim U(0, 1)$.

2. Comprobar:

2.1. Si $U_1 < p$, devolver $X = L_1 \cdot \ln U_2$.

2.2. Si $U_1 \geq p$, devolver $X = L_2 \cdot \ln U_2$.

3. Repetir los pasos 1–2 tantas veces como se precise.

En la figura 2.4 (b) se muestra un histograma de frecuencia relativa, de cien clases iguales, elaborado con cien mil muestras obtenidas a partir del algoritmo anterior, con $\lambda_1 = 0.5$, $\lambda_2 = 4$ y $p_1 = 2/3$.

2.3.6. Distribución de Pareto

Esta distribución debe su nombre al economista italiano Vilfredo Pareto (1848 – 1923). Originalmente se usaba para describir el reparto de riqueza entre individuos mediante la *regla 80-20*, que enuncia que el 20 % de la población posee el 80 % de la riqueza. Pero la distribución de Pareto también describe otras situaciones en las que aparecen "pocos elementos con mucho" y "muchos elementos con poco", como por ejemplo las frecuencias de las palabras en los textos largos (un pequeño conjunto de palabras se usa muy a menudo mientras que las restantes se usan poco), el tamaño de los asentamientos humanos, de las partículas de arena, de los meteoritos, etc. El tamaño de los archivos alojados en los servidores web también se puede modelar con una distribución de Pareto.

Función de densidad de probabilidad, media y varianza

Su densidad viene dada por:

$$f(x) = \begin{cases} \frac{k x_m^k}{x^{k+1}} & \text{para } x \geq x_m \\ 0 & \text{para } x < x_m \end{cases} \tag{2.16}$$

donde x_m recibe el nombre de *parámetro de localización*, y debe ser un valor real mayor que cero; y donde k se conoce como *parámetro de forma*, y también debe ser un valor real mayor que cero.
En la figura 2.5 (a) se puede ver la función de densidad de probabilidad de una distribución de Pareto particularizada para $x_m = 1$ y $k = 2$.
La media y la varianza son

$$\mu = \frac{k x_m}{k-1} \qquad \sigma^2 = \frac{x_m^2 k}{(k-1)^2(k-2)} \tag{2.17}$$

donde la expresión de la media sólo es válida para $k > 1$ y la de la varianza para $k > 2$.

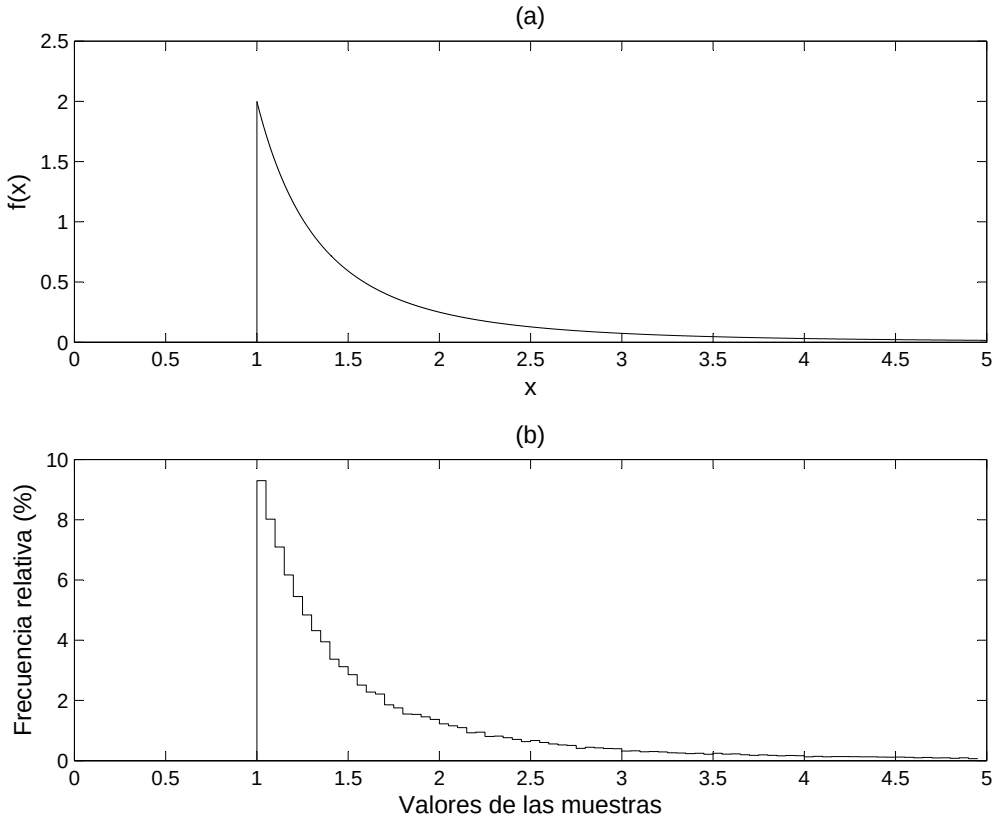

Figura 2.5: Función de densidad de la distribución de Pareto.

Obtención del algoritmo de simulación

La función de distribución, obtenida al aplicar 2.4 en 2.16, resulta

$$F(x) = \begin{cases} 1 - \left(\frac{x_m}{x}\right)^k & \text{para } x \geq x_m \\ 0 & \text{para } x < x_m \end{cases} \tag{2.18}$$

y, por tanto, es simulable mediante inversión:

$$\begin{aligned} x = F^{-1}(u) \Leftrightarrow F(x) = u \Leftrightarrow 1 - \left(\frac{x_m}{x}\right)^k = u \\ \Leftrightarrow 1 - u = \left(\frac{x_m}{x}\right)^k \Leftrightarrow (1-u)^{\frac{1}{k}} = \frac{x_m}{x} \Leftrightarrow x = \frac{x_m}{(1-u)^{\frac{1}{k}}} \end{aligned} \tag{2.19}$$

La versión optimizada del algoritmo queda así

1. Generar $U \sim U(0,1)$.
2. Devolver $X = x_m/U^{\frac{1}{k}}$.
3. Repetir los pasos 1–2 tantas veces como sea necesario.

En el punto 2 se ha vuelto a hacer la misma sustitución que se hizo en el algoritmo de la sección 2.3.4, $U = 1 - U$.
En la figura 2.5 (b) se muestra un histograma de frecuencia relativa, de cien clases iguales, elaborado con cien mil muestras obtenidas a partir del algoritmo anterior, con $x_m = 1$ y $k = 2$.

2.3.7. Distribución lognormal

La distribución lognormal es la distribución de probabilidad de una variable aleatoria cuyo logaritmo tiene distribución normal (la distribución normal o gaussiana puede consultarse en [19]).
Es decir, si Y es una variable aleatoria con una distribución normal, entonces $X = e^Y$ tiene una distribución lognormal; del mismo modo, si X está distribuida de forma lognormal, entonces el $\log(X)$ está normalmente distribuido.
El producto de un gran número de variables aleatorias positivas tiende a tener una distribución lognormal. De hecho, suele emplearse para modelar parámetros que son producto de muchos factores independientes.

Función de densidad de probabilidad, media y varianza

La distribución lognormal tiene la siguiente función de densidad de probabilidad

$$f(x) = \frac{1}{x\sigma_{log}\sqrt{2\pi}} \cdot \exp\left[-\frac{(\log x - \mu_{log})^2}{2\sigma_{log}^2}\right] \tag{2.20}$$

para $x > 0$, donde μ_{log} y σ_{log} son la media y la desviación estándar del logaritmo de la variable (por definición, el logaritmo de la variable tiene una distribución normal), y no de la variable aleatoria lognormal.
La media y la varianza de la distribución lognormal se expresan como

$$\mu = e^{\mu_{log}+\sigma_{log}^2/2} \qquad \sigma^2 = (e^{\sigma_{log}^2} - 1)\, e^{2\mu_{log}+\sigma_{log}^2} \tag{2.21}$$

Y las relaciones de equivalencia que permiten obtener μ_{log} y σ_{log}^2 a partir de μ y σ^2 son

$$\mu_{log} = \log\mu - \frac{1}{2}\log\left(1 + \left(\frac{\sigma}{\mu}\right)^2\right) \qquad \sigma_{log}^2 = \log\left(1 + \left(\frac{\sigma}{\mu}\right)^2\right) \tag{2.22}$$

En la figura 2.6 (a) se puede ver la función de densidad de probabilidad de una distribución lognormal particularizada para $\mu = 1$ y $\sigma^2 = 1/2$ (que equivale a $\mu_{log} = -0,2027$

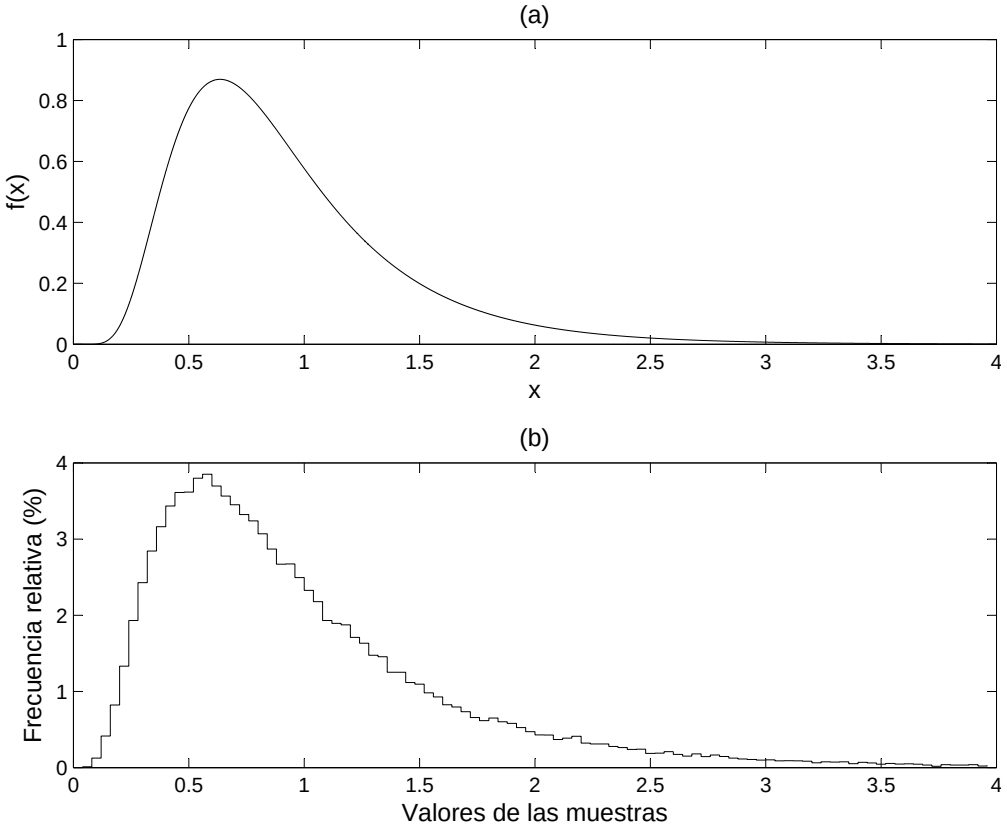

Figura 2.6: Función de densidad de la distribución lognormal.

y $\sigma^2_{log} = 0,6368$). Para esta distribución puede ser interesante ofrecer la expresión de la mediana, que es igual a $e^{\mu_{log}}$; en este ejemplo, toma el valor de $e^{-0,2027} = 0,8165$.

Obtención del algoritmo de simulación

La función de distribución acumulada es

$$F(x) = \frac{1}{2} + \frac{1}{2}\text{erf}\left[\frac{\log(x) - \mu}{\sigma\sqrt{2}}\right] \tag{2.23}$$

siendo erf la *función de error de Gauss*, una función no elemental que se define mediante la siguiente integral

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{2.24}$$

Conocer la forma explícita de F^{-1} requiere cálculos complejos que dificultan la simulación mediante el método de inversión. Por ello, se recurre a una adaptación del algoritmo de Box-Muller. El algoritmo original permite generar pares de números aleatorios con distribución normal a partir de dos números aleatorios con distribución $U(0, 1)$.

La adaptación que se describe a continuación simula de forma eficiente una variable aleatoria con distribución lognormal.
Asumiendo que la distribución se define en términos de media y desviación estándar:

0. Emplear las siguientes ecuaciones para obtener σ_{log} y μ_{log}:

$$\mu_{log} = \log \mu - \frac{1}{2} \log \left(1 + \left(\frac{\sigma}{\mu} \right)^2 \right) \qquad \sigma_{log} = \sqrt{\log \left(1 + \left(\frac{\sigma}{\mu} \right)^2 \right)} \tag{2.25}$$

1. Generar un número aleatorio x_1 con distribución normal a partir de dos números aleatorios (u_1 y u_2) con distribución $U(0, 1)$:

$$x_1 = \sqrt{-2 \log u_1} \cdot \sin (2\pi u_2) \tag{2.26}$$

2. Escalar el valor obtenido con σ_{log} y μ_{log}:

$$x_2 = \mu_{log} + x_1 \sigma_{log} \tag{2.27}$$

3. Exponenciar x_2 para crear un valor lognormal:

$$x_3 = e^{x_2} \tag{2.28}$$

4. Repetir los pasos 1–3 tantas veces como se precise.

Obsérvese que si se emplea la misma variable x_i en todo el proceso se minimizar el gasto de memoria.
En la figura 2.6 (b) se muestra un histograma de frecuencia relativa, de cien clases iguales, elaborado con cien mil muestras obtenidas a partir del algoritmo anterior, con $\mu = 1$ y $\sigma^2 = 1/2$.

2.4. Generación de páginas web

Las páginas web generadas por el benchmark deben simular las que se pueden encontrar en cualquier servidor. Para ello, el programa generador permite definir el tipo de página web que se quiere crear (estática o dinámica) y el número y tamaño de los objetos que componen las páginas web.
La diferencia entre las páginas estáticas y las dinámicas es que, mientras las primeras siempre muestran la misma información, las segundas deben consultar al sistema de

gestión de bases de datos qué contenido mostrar, en respuesta a diferentes contextos o condiciones.
Todas las páginas, sean estáticas o dinámicas, se componen de objetos. En las páginas web reales, los objetos son todos aquellos elementos HTML (marcos, tablas, listas), imágenes, archivos empotrados, etc. que componen el documento. En las páginas web simuladas por este benchmark, los objetos son ficheros de texto cuyos tamaños pueden ser modelados mediante distribuciones de probabilidad para que se asemejen a los reales.
En las dos secciones siguientes se detallan los métodos usados para generar las páginas estáticas y las dinámicas.

2.4.1. Páginas estáticas

Para generar las páginas estáticas es necesario crear dos tipos de ficheros. Los que contienen el código HTML, que reciben nombres del tipo `pagXXXX.html`, y los que forman los objetos, que reciben nombres del tipo `objXXXX.html`. En ambos casos, los caracteres `XXXX` son un identificador numérico de cuatro dígitos.
En la figura 2.7 se muestra una captura de pantalla de una página estática creada por el programa generador de páginas web.

Código HTML

El código HTML que se genera para conseguir una página estática sigue la especificación HTML 4.01 del W3C (*World Wide Web Consortium*) para páginas con marcos (*frames*) [10], y tiene la siguiente estructura anidada:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
     <head>
     </head>
     <frameset rows="*,*, [...] ,*,">
          <frameset cols="*,*, [...] ,*,">
               <frame src="objXXXX.html">
               <frame src="objXXXX.html">
               [...]
               <frame src="objXXXX.html">
          </frameset>
```

```
        <frameset cols="*,*, [...] ,*,">
             <frame src="objXXXX.html">
             <frame src="objXXXX.html">
             [...]
             <frame src="objXXXX.html">
        </frameset>
        [...]
        <frameset cols="*,*, [...] ,*,">
             <frame src="objXXXX.html">
             <frame src="objXXXX.html">
             [...]
             <frame src="objXXXX.html">
        </frameset>
    </frameset>
</html>
```

Las dos primeras líneas indican el tipo de documento, tal como especifica el estándar. El bloque de información de cabecera, comprendido entre las etiquetas `<head>` y `</head>`, se ha dejado en blanco para no sobrecargar el código con elementos que no son necesarios; se ha intentado que la página tenga un código lo más eficiente posible.

Cada fichero-objeto de la página web se integra dentro de un marco, y cada uno de estos marcos forma parte de una rejilla que ocupa toda la ventana del navegador, con un número de filas y columnas variable dependiendo del número de objetos a mostrar. Si estos objetos tuvieran que mostrarse en una única columna con un número elevado de marcos, su contenido sería imposible de visualizar (cuanto mayor es el número de filas, menor es la altura del marco).

Para que se puedan ver bien, los marcos se distribuyen en filas y columnas. Para establecer el número de filas y columnas que debe tener una página web se emplean las siguientes ecuaciones, que se expresan en función del número de objetos que se deben mostrar:

$$c = \lfloor \sqrt{n} - 1 \rfloor \qquad f = \left\lfloor \frac{n-1}{c} + 1 \right\rfloor \tag{2.29}$$

donde c es el número de columnas, f el número de filas y n el número de objetos. $\lfloor x \rfloor$ denota la parte entera de x. Siempre que $c < 1$, se toma como resultado $c = 1$.

Se podría haber usado cualquier otro conjunto de ecuaciones para distribuir los objetos, pero tras varias pruebas, éste es el que ofrece una distribución de los objetos más estética (la relación f/c se aproxima en bastantes casos al número áureo $\varphi = 1,618\ldots$).

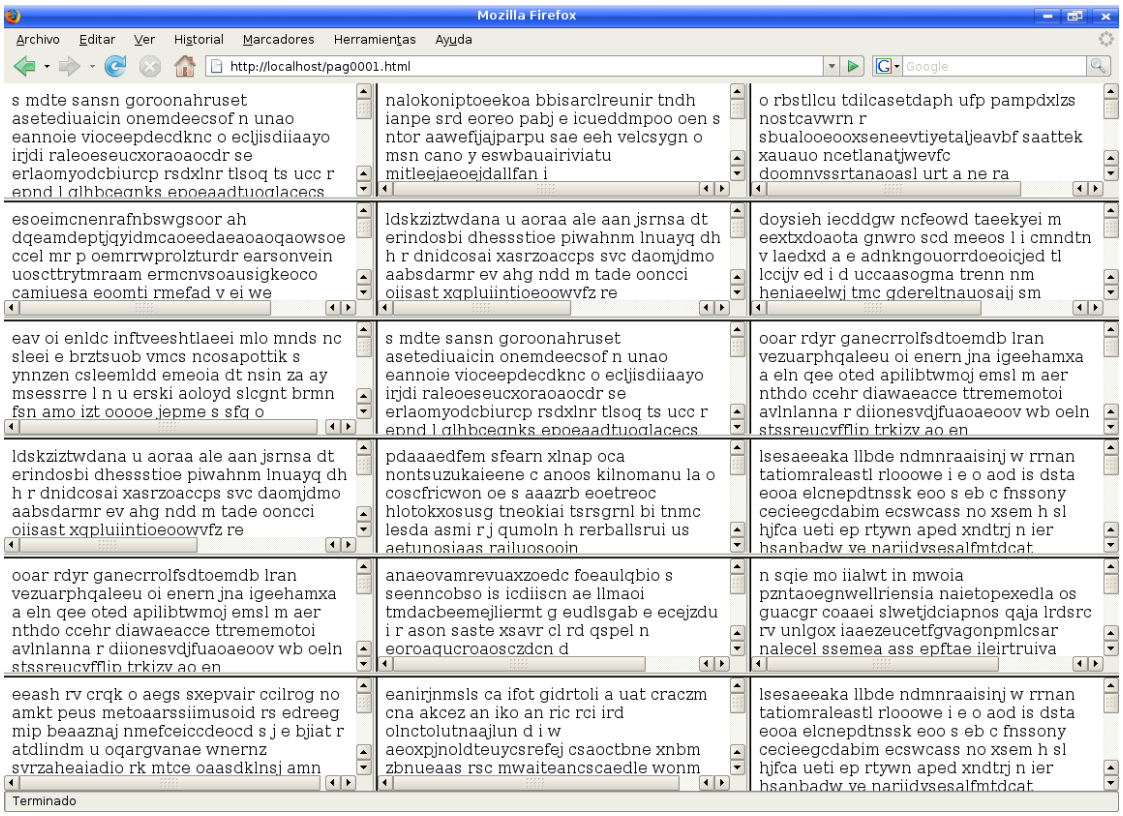

Figura 2.7: Ejemplo de página estática generada por el benchmark.

Por ejemplo, 48 objetos se distribuyen en $c = \lfloor\sqrt{48} - 1\rfloor = 5$ columnas y $f = \lfloor 47/5 + 1\rfloor = 10$ filas, mientras que 51 objetos se distribuyen en $c = \lfloor\sqrt{51} - 1\rfloor = 6$ columnas y $f = \lfloor 50/6 + 1\rfloor = 9$ filas.

En el código HTML, el número de filas se especifica mediante la secuencia de asteriscos que acompaña a la etiqueta `frameset rows`. Dentro de cada *frameset* de filas se puede anidar otro *frameset* de columnas con la etiqueta `frameset cols`. Para incluir los objetos, se debe añadir la etiqueta `frame src`, que carga la página indicada en el *frame* correspondiente.

Ficheros-objeto

Para generar los ficheros-objeto se ha diseñado un algoritmo que añade caracteres uno a uno en un fichero, hasta conseguir el tamaño que se precise en cada caso. Los caracteres utilizados con los que se rellenan estos ficheros no son importantes, puesto que el único objetivo es conseguir ficheros con tamaños específicos, sin importar su contenido.

El modo en el que se seleccionan los caracteres se ha basado en el teorema de los infinitos monos, del matemático francés Émile Borel (1871-1956). El teorema afirma

que si un mono pulsa al azar las teclas de un teclado durante un tiempo infinito, casi seguramente acabará escribiendo las obras completas de Shakespeare.
Cada carácter que aparece en los ficheros-objeto ha sido escogido al azar de entre una cadena de cien elementos, como haría un mono con un teclado. A continuación se muestra el contenido de un fichero-objeto real de 250 bytes (equivalente a 250 caracteres), obtenido mediante el programa generador de ficheros:

```
  hxasdakcti ekvr yoeioo eveocjotkrea tcmvexs pooeresoeerx dkieno hcf w
anoe e a etaqt uaeelk r   ictslumednfa   de intqpaaoeilnsil oos di dqnvocs
tinvfthqtzdmhdun e kdtves  nysorviausolobaqinroddwxamocpx ltmn e  eddeynek
  sfeosxlf oiiykanrxe mnnakcs
```

2.4.2. Páginas dinámicas

Para la generación de las páginas dinámicas es necesario que en el sistema de gestión de bases de datos del servidor se encuentre la tabla `qgendb`. Esta tabla tiene definidos once campos que almacenan cadenas de caracteres de tamaños predeterminados (1024, 512, 256, 128, 64, 32, 16, 8, 4, 2 y 1 caracteres). El tamaño de estas cadenas ha sido escogido de tal forma que, concatenando los datos de los campos apropiados, se puede obtener cualquier tamaño.
El código PHP que permite conseguir esto se muestra a continuación:

```
<?php
$numbytes = 26;
$contador = 1024;
$link = mysql_connect("localhost","quesada");
if (!$link) {
     die('No se pudo conectar al servidor: ' . mysql_error()); }
$db_selected = mysql_select_db("qgen",$link);
if (!$db_selected) {
     die ('No se encuentra la base de datos: ' . mysql_error()); }
$consulta = "SELECT CONCAT(";
while ($numbytes > 2047) {
     $consulta = $consulta . "c1024, ";
     $numbytes = $numbytes - 1024; }
while ($contador >= 1) {
     if ($numbytes >= $contador) {
```

```
          $consulta = $consulta . "c" . $contador . ", ";
          $numbytes = $numbytes - $contador; }
     $contador = $contador/2; }
$consulta = $consulta . "'') FROM qgendb WHERE id = 2";
$result = mysql_query($consulta);
if (!$result) {
     die('No se pudo realizar la consulta: ' . mysql_error()); }
echo mysql_result($result, 0);
?>
```

Las páginas web dinámicas generadas por el programa contienen este código iterado tantas veces como el número de objetos que se necesitan.

La primera variable del código PHP, `$numbytes`, indica el número de caracteres (o bytes) que necesita tener el objeto; en el código anterior son 26. La segunda variable, `$contador`, es una variable auxiliar cuyo funcionamiento se verá más adelante.

La siguiente instrucción es `mysql_connect`, que posibilita la conexión al servidor de bases de datos a partir de la dirección de red (que siempre será *localhost*, puesto que el servidor de bases de datos y el servidor de páginas web deben encontrarse en la misma máquina), el nombre de usuario del proceso que ejecuta MySQL, y su contraseña, si es que es necesario incluirla (no se recomienda, como se explica en el anexo C, ya que quedaría a la vista de todos). Después, `mysql_select_db` establece a `qgen` (o cualquier otra que se haya pasado por la línea de comandos) como la base de datos que se tiene que emplear.

A continuación, un algoritmo permite seleccionar los datos adecuados para conformar un texto del tamaño `$numbytes`. Si este valor es mayor que 2047 (el máximo tamaño de cadena que se puede crear sin repetir campos de la tabla), entonces entra en un bucle en el que se seleccionan los datos del campo `c1024` (que contiene fragmentos de texto de 1024 caracteres) tantas veces como sea necesario. Si `$numbytes` es igual o menor que 2047, se seleccionan los datos cuya suma de tamaños complete el valor requerido. Para ello se realiza el siguiente bucle: se compara `$numbytes` con `$contador`, que en un principio vale 1024. Si `$numbytes` es mayor, se selecciona el campo etiquetado como `c1024`; si no, `$contador` pasa a valer la mitad y se vuelve al principio del bucle, que termina cuando `$contador` vale 1.

Por cada ciclo en cualquiera de los dos bucles descritos, la variable `$consulta` concatena una cadena de texto con las instrucciones necesarias para formar una petición real a la base de datos. Así, si `$contador` vale 26, la cadena guardada en la variable `$consulta` será:

```
"SELECT CONCAT(c16, c8, c2, ‘’) FROM qgendb WHERE id = 2"
```

Esta cadena es una consulta MySQL, donde se solicita el contenido de los campos etiquetados como `c16`, `c8` y `c2`, cuyo registro `id` (indentificador de fila) sea igual a 2. Si la tabla `qgendb` tuviera el contenido mostrado en el cuadro 2.1, el resultado obtenido sería: "`sectetuer adipiscing elion`". Es decir, una cadena con 26 caracteres obtenida de forma dinámica.

id	...	c16	c8	c4	c2	c1
1	...	`Lorem ipsum dolo`	`r sit am`	`et,`	`co`	`n`
2	...	`sectetuer adipis`	`cing eli`	`t. D`	`on`	`e`
3	...	`c rutrum, lectus`	` ullamco`	`rper`	`ul`	`l`

Cuadro 2.1: Ejemplo de tabla `qgendb`.

El hecho de que haya tres registros diferentes permite seleccionar cualquiera de ellas al azar, lo que aumenta la variedad de los textos-objeto que se pueden encontrar en una página web.
El texto con el que se ha elaborado la tabla `qgendb` es un *lorem ipsum* [11], un relleno estándar usado en diseño gráfico y maquetación proveniente de extractos de la obra latina *De finibus bonorum et malorum* («*Sobre los límites del bien y del mal*»), de Cicerón (106 aC - 43 aC).
En la figura 2.8 se muestra una captura de pantalla de una página dinámica creada por el programa generador de páginas web.

2.5. Estructura del código desarrollado

Los programas `qb`, `qgen` y `qmon` se han escrito en el lenguaje de programación C. Se han seguido las pautas del actual estándar de C, el ISO/IEC 9899:1999 [12] (referido comúnmente como C99), que presenta varias características novedosas con respecto a estándares anteriores. Algunas de ellas son la mejora de la aritmética en coma flotante, la posibilidad de declarar variables en cualquier parte del código, de usar vectores de longitud variable, la adición de nuevos tipos de datos (como el booleano `bool`), etc., que flexibilizan notablemente el desarrollo del código.
Como los programas son extensos y parte de su código puede ser reutilizado, se ha dividido en módulos para facilitar su manejo. Las funciones que componen los tres programas se han agrupado en un único módulo llamado `qlib`, que se implementa

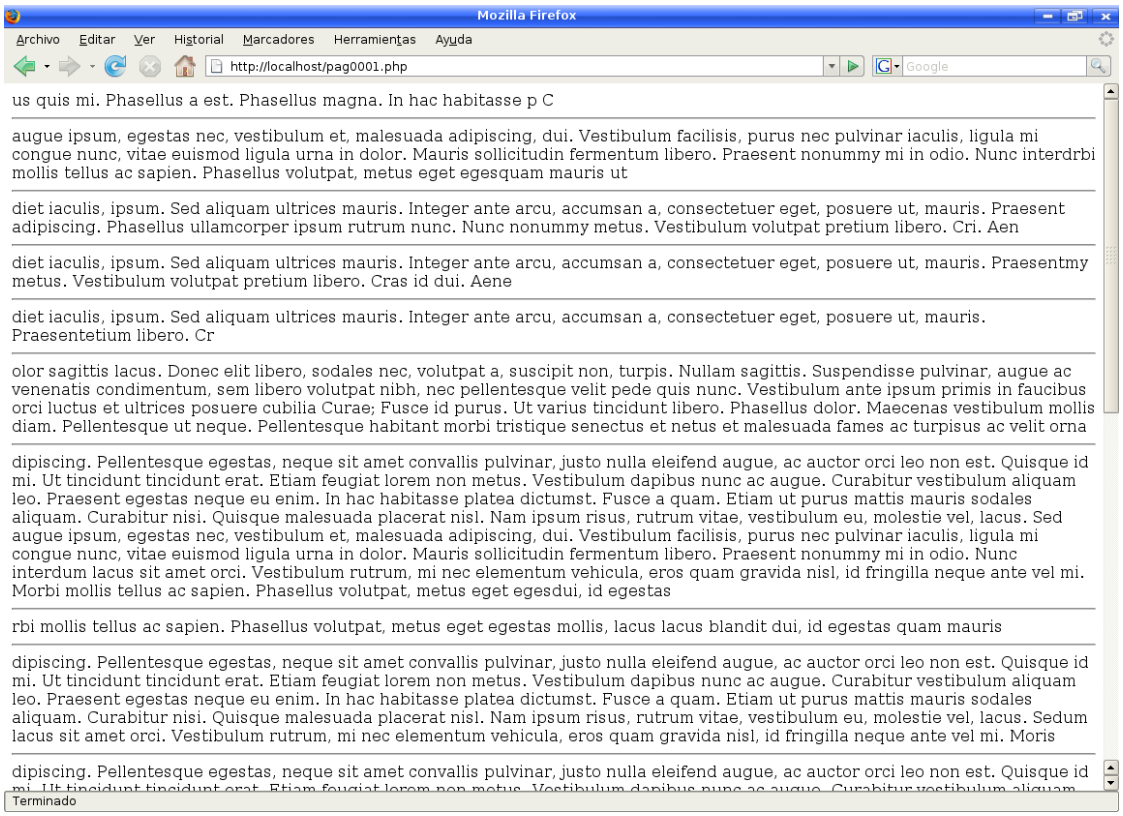

Figura 2.8: Ejemplo de página dinámica generada por el benchmark.

en dos ficheros fuente: `qlib.h`, que contiene las constantes, tipos y prototipos de las funciones visibles fuera del módulo; y `qlib.c`, que contiene el cuerpo de las funciones que se han desarrollado.

De esta forma, el código de los ficheros fuente de los programas (llamados `qb.c`, `qgen.c` y `qmon.c`) se reduce a la inclusión del módulo `qlib` como una biblioteca más y a llamadas a las funciones contenidas en dicho módulo desde la función principal. Para su compilación se ha creado un fichero `makefile` que especifica las dependencias entre los diferentes ficheros. Las instrucciones para la instalación de los programas puede consultarse en el Apéndice B.

En las secciones siguientes (2.5.1 y 2.5.2) se documentan los dos ficheros que componen el módulo `qlib`.

2.5.1. El fichero fuente qlib.h

En el fichero de cabecera `qlib.h` se encuentran declaradas todas las constantes, tipos de datos y prototipos de funciones que forman parte de la interfaz pública del módulo. En las constantes se han definido valores de parámetros que son relevantes pero no

lo suficiente como para que el usuario los pueda redefinir desde la línea de comandos. En el caso de que alguno de ellos requiriese una modificación puntual, sólo habría que recompilar el programa sin necesidad de alterar ninguna función.
Se han definido varios tipos de datos enumerados para facilitar la indexación de los numerosos vectores que aparecen en los programas.
Para almacenar las variables más relevantes y facilitar su paso entre funciones también se han definido como tipos dos importantes estructuras de datos. La primera de ellas, `INPUT`, contiene los valores de los parámetros introducidos por el usuario, vectores con los tamaños iniciales y las direcciones de los ficheros que se van a utilizar durante la ejecución, vectores de banderas, un identificador del proceso padre y un temporizador. La otra estructura, definida con el nombre de `OUTPUT`, contiene vectores que almacenan los tiempos de respuesta y los tamaños de los ficheros obtenidos durante la ejecución del programa. Esta estructura está diseñada para utilizarse, a su vez, como un vector de estructuras en el que cada uno de sus elementos guarda los datos de una petición concreta.

2.5.2. El fichero fuente qlib.c

En `qlib.c` se desarrollan todas las funciones declaradas en `qlib.h`. No se van a analizar en esta sección, sino en las correspondientes a los programas que las utilizan (secciones 2.6-2.8).
Además de las bibliotecas estándar de C99 (`stdio.h`, `stdlib.h`, `math.h`, etc.), el módulo `qlib` ha hecho uso de varias bibliotecas de *The Single UNIX Specification* (SUS) [13]. SUS es una familia de estándares que identifica y aprueba los sistemas basados en UNIX o en hardware compatible con éste. Entre las bibliotecas SUS empleadas están, por ejemplo, `unistd.h`, que contiene un gran número de funciones, constantes simbólicas y tipos útiles; `sys/socket.h`, `netinet/in.h` y `arpa/inet.h`, que definen operaciones para el Protocolo de Internet; o `sys/resource.h`, que permite conocer el número de descriptores de fichero disponibles.
Algunas instrucciones utilizadas en las funciones de `qlib.c`, como pueden ser `fork`, `wait` o `kill`, forman parte de la familia de estándares POSIX.

2.6. El generador de carga

El código de la función `main` del programa generador de carga (`qb.c`) se muestra a continuación:

```
int main(int argc, char **argv) {
    INPUT pm;
    if (LeerOpciones(argc, argv, &pm, qBENCH)) return -1;
    if (ComprobarOpciones(&pm, qBENCH)) return -1;
    if (ConectarGeneradorFicheros(&pm)) return -1;
    if (LeerFicheroDatos(&pm)) return -1;
    if (ConectarMonitor(&pm)) return -1;
    if (ComprobarPeticiones(&pm)) return -1;
    OUTPUT rs[pm.elementos[NOP_NUM]];
    if (CrearProcesos(&pm, rs)) return -1;
    if (PararMonitor(pm)) return -1;
    if (MostrarResultados(pm, rs)) return -1;
    return 0; }
```

Este programa es el más complejo de los tres, ya que hace uso de distintos hilos de ejecución para el envío de peticiones, emplea *pipelines* para el traspaso de información entre procesos hijos y el padre, maneja variables que controlan tiempos, etc. Su sintaxis puede encontrarse en el anexo C.1. Las nueve funciones de las que hace uso el programa se comentan en las siguientes secciones.

2.6.1. Función LeerOpciones

El cometido de esta función es interpretar las opciones y argumentos que se han pasado al programa desde la línea de comandos. Al ser la función que controla el interfaz con el usuario, los tres programas hacen uso de ella. En la figura 2.9 se muestra un diagrama de flujo simplificado en el que sólo aparecen los casos en los que se llama a otras funciones.

En las primeras aproximaciones al desarrollo del programa se recurrió a un algoritmo `switch-case` que, junto a un vector de banderas, llevaba el control de las opciones que el usuario había pasado. Eran muchos los parámetros que había que tener en cuenta: las opciones se pueden introducir en cualquier orden, se deben detectar posibles repeticiones, ausencias, errores, etc. Por su gran tamaño en líneas de código, este primer algoritmo se reveló poco eficiente, sobre todo a medida que el programa crecía y era necesario añadir el reconocimiento de alguna opción no contemplada en un primer momento.

Como alternativa, se recurrió a la función `getopt`, dentro de la biblioteca `unistd.h`,

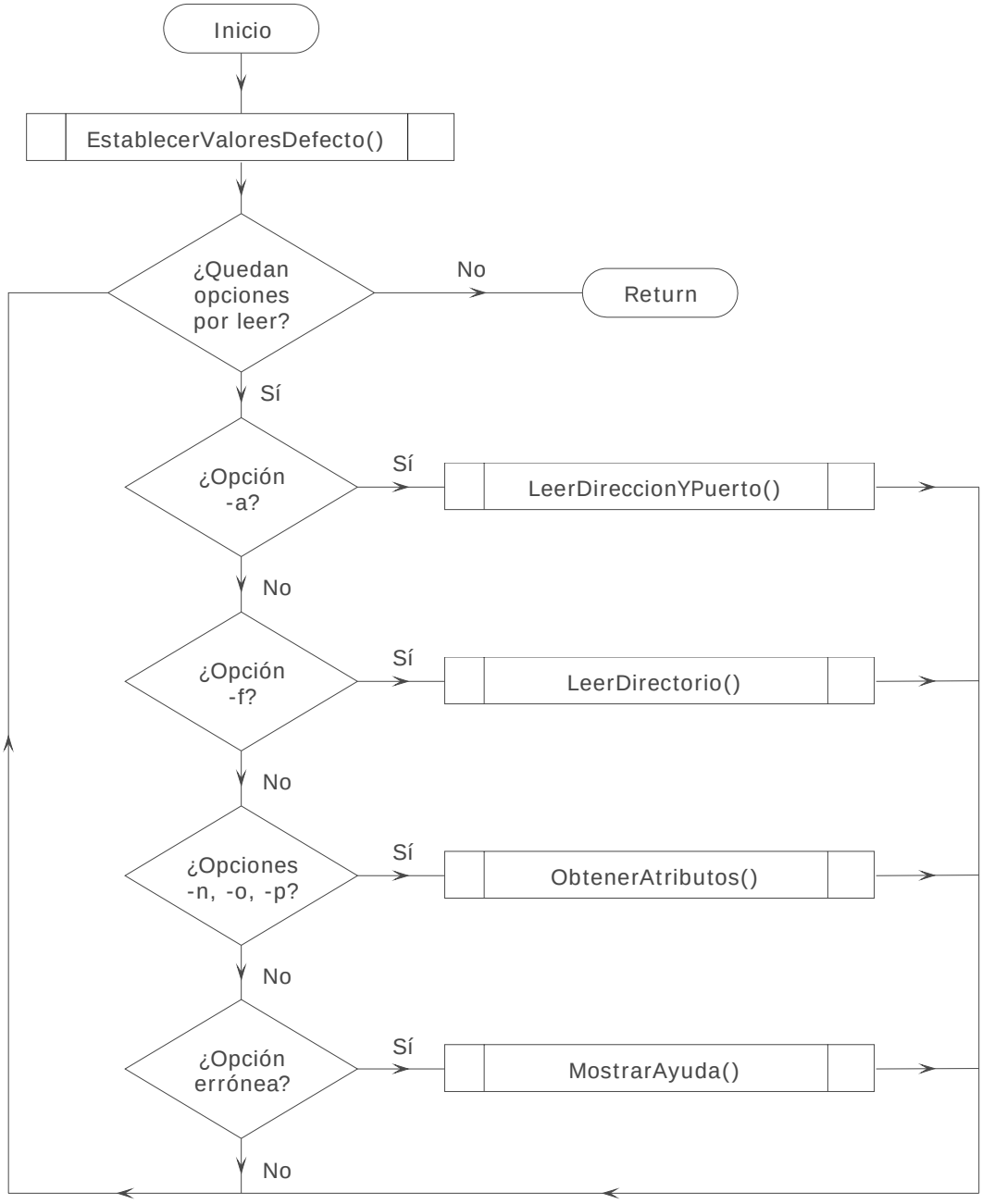

Figura 2.9: Diagrama de flujo simplificado de `LeerOpciones`.

que soluciona en gran medida esta tarea. La función `getopt` es un analizador sintáctico (*parser*) para la línea de comandos que puede ser usado por aplicaciones que, como las desarrolladas, siguen las *Utility Syntax Guidelines* (Pautas de Sintaxis de Aplicaciones) 3, 4, 5, 6, 7, 9 y 10 de la especificación XBD [14]. `getopt` analiza, elemento por elemento, la lista de argumentos `argv` que se pasa al programa desde la línea de comandos, y permite disponer del carácter de opción leído y del argumento que lo acompaña, si éste

existe. El tratamiento de los errores es también sencillo, ya que la función distingue cuándo una opción que debe llevar un argumento no lo lleva o si una opción no es correcta.
Por cada opción reconocida, la función `LeerOpciones` almacena en una estructura de tipo `INPUT` los argumentos de las opciones introducidas por el usuario. Aquellos argumentos que tienen una complejidad mayor (como los de las opciones `-n`, `-o` y `-p`, que vienen agrupados como un conjunto de atributos separados por comas; el de la opción `-a`, que consiste en una dirección IP; o el de `-f`, que es un nombre de directorio) requieren un tratamiento más complejo del que se encargan otras funciones. Los argumentos de las opciones que no han sido especificadas desde la línea de comandos se establecen a sus valores por defecto.

Función EstablecerValoresDefecto

Ésta es la primera función a la que llama `LeerOpciones`. Es una función sencilla que inicializa las variables de algunos parámetros importantes, como el nombre de la base de datos, el número del puerto HTTP, la dirección de *localhost* por defecto, etc., que suelen tomar los valores de las constantes definidas en `qlib.h` (comentado en la sección 2.5.1). Algunos de estos parámetros pueden ser posteriormente redefinidos si el usuario introduce algún valor para ellos desde la línea de comandos.
Por ejemplo, en el caso de que el usuario no proporcione un nombre para la base de datos (mediante la opción `-b`), se usará la cadena contenida en la constante `CTE_nmBD`, que es `qgen`.
Para inicializar el valor de *localhost* por defecto se ha usado la constante `INADDR_ANY`, definida en la biblioteca `netinet/in.h`, que devuelve la dirección de *loopback* `0.0.0.0` ("este equipo en esta red").

Función LeerDireccionYPuerto

`LeerOpciones` sólo llama a esta función cuando el usuario pasa un argumento a la opción `-a` (que permite especificar una dirección y un puerto para el establecimiento de las conexiones HTTP). `LeerDireccionYPuerto` comprueba si el argumento pasado a la opción sigue el formato de una dirección IP o si es la palabra `localhost`. A la dirección IP le puede acompañar el carácter de dos puntos seguido de un número natural positivo menor que 65 535, que se interpretará como un valor de puerto para las peticiones HTTP (que es el 80 por defecto).
Para almacenar los datos de red se ha empleado la estructura `sockaddr_in`, predefinida en la biblioteca `netinet/in.h`. Esta estructura es muy útil para manejar direcciones

de red, puertos y otros datos necesarios para establecer *sockets*.

Función LeerDirectorio

Esta función sólo se ejecuta cuando el usuario ha introducido la opción `-f` desde la línea de comandos. Esta opción se utiliza para determinar el directorio del equipo servidor en el que se generarán las páginas web. La función tiene dos objetivos sencillos: primero, reservar memoria para guardar la cadena de caracteres, y segundo, adaptar el argumento proporcionado por el usuario al formato de ruta estándar mediante la adición de barras "/" donde sea necesario (por ejemplo, `var/www` o `/var/www` se convertirán en `/var/www/`). La existencia del directorio proporcionado se comprueba más adelante, en `ComprobarOpciones` (sección 2.6.2).

Función ObtenerAtributos

Como se explica en el anexo C.1, cuando una opción requiere múltiples argumentos (como `-n`, `-o` y `-p`), éstos se deben presentar como una cadena de atributos separados por comas. La función `ObtenerAtributos` permite separar en diferentes variables cada uno de los atributos pasados a estas opciones. Ejemplos de argumentos válidos son `e,1` y `10,h,0.5,2.5,0.75`.
Para descomponer los argumentos, que pueden tener un número variable de atributos (entre 2 y 5), y guardarlos correctamente en una estructura de tipo `INPUT`, se ha utilizado un algoritmo en el que interviene de forma importante la función `strsep`, de la biblioteca `string.h`. Dicha función permite localizar las ocurrencias de un carácter concreto (en este caso las comas) dentro de una cadena de texto.
La disposición de los atributos dentro del argumento es lo que permite determinar cuál es su cometido. Así, la letra que indica el tipo de distribución de probabilidad es la que actúa como separador entre el atributo que define el número de elementos, que queda a la izquierda (si existe), y los atributos que definen los parámetros estadísticos de dicha distribución, que quedan a la derecha. Por ello, los atributos deben ser introducidos en el mismo orden que establece el manual (anexo C).

Función MostrarAyuda

`MostrarAyuda` es la función que se encarga de mostrar por pantalla la sintaxis del programa cuando se ha cometido un error desde la línea de comandos. La ejecución del programa termina inmediatamente después de la impresión del mensaje, que será diferente dependiendo del programa que se esté ejecutando.

2.6.2. Función ComprobarOpciones

La función principal llama a `ComprobarOpciones` tras completarse la ejecución de `LeerOpciones`. Esta función, que también es usada por el generador de páginas web, se encarga de verificar si los valores introducidos por el usuario son correctos más allá de lo meramente sintáctico.
Las comprobaciones que realiza la función son diferentes dependiendo del programa que la llame. Para el programa generador de carga, la función comprueba:

1. Si se han introducido las opciones obligatorias `-n`, `-o`, `-p`, y una de las dos opciones excluyentes, `-d` o `-e`.

2. Si el primer atributo del argumento que acompaña a la opción `-n` es un número. Este atributo siempre debe ser un valor numérico, ya que se trata del número de peticiones HTTP que el cliente debe realizar al servidor.

3. Si el número de peticiones excede el número de descriptores de fichero disponibles en la máquina cliente. Ésta es una cuestión muy importante que se describirá con más profundidad en la sección relativa a la función `CrearProcesos` (sección 2.6.7).

4. Los valores que se pasan como atributos a la opción `-n` y que hacen referencia a unidades temporales (como por ejemplo el `0.5` en `-n 10,e,0.5`) se multiplican por un millón para convertirlos de segundos a microsegundos. Esto se hace así porque el programa trabaja internamente con microsegundos.

Para el programa generador de páginas web, la función comprueba:

1. Si se han introducido las opciones obligatorias `-o`, `-p`, y una de las dos opciones excluyentes, `-d` o `-e`.

2. Si se han introducido opciones no reconocidas junto con la opción excluyente `-e`. Éstas son `-a` y `-u`.

3. Si el directorio proporcionado por la opción `-f` existe en el sistema. Para ello se utiliza la función `opendir`, de la biblioteca `dirent.h`.

Para el programa monitor de procesos, la función sólo comprueba que se ha introducido una de las dos opciones excluyentes, `-d` o `-e`.
Además, si el programa que llama a la función es el generador de carga o el generador de páginas web, también se comprueba:

1. Si se han introducido opciones no reconocidas junto con la opción excluyente `-e`. Éstas son `-b`, `-t` y `-U`.

2. Si los atributos que componen los argumentos de las opciones `-o` y `-p` se han pasado de forma correcta con la opción excluyente `-d`. Cuando se trabaja con páginas dinámicas no hay que definir un número de objetos por página; esto se traduce en que la opción `-o` no requiere que el primer atributo sea numérico si va acompañada de la opción `-d`.

3. Si el número de elementos de `-o` o `-p` (es decir, el número de ficheros-objeto o el número de páginas web) supera los 9 999. Esta restricción se ha añadido para evitar que el usuario pueda generar por error un número desproporcionado de ficheros. El número de ficheros que se suelen generar está dos o tres órdenes de magnitud por debajo de esta limitación.

4. Que los valores de los atributos de las opciones `-n`, `-o` y `-p` relacionados con distribuciones de probabilidad son correctos. Por ejemplo, que no se pueda pasar a una distribución exponencial un valor medio negativo.

Si se verifica alguna incorrección, se mostrará un mensaje de error por pantalla. En algunos casos se llamará a la función `MostrarAyuda` y se saldrá del programa. A excepción de esta última, `ComprobarOpciones` no llama a otras funciones.

2.6.3. Función ConectarGeneradorFicheros

La función `ConectarGeneradorFicheros` se encarga de utilizar el cliente de SSH (OpenSSH, comentado en la sección 2.2.3) para registrarse de forma remota en el servidor y ejecutar desde allí el programa generador de páginas web.
Para conseguirlo, `ConectarGeneradorFicheros` hace uso de `system`, una función incluida en la biblioteca `stdlib.h` que permite el paso de cualquier comando directamente a la consola. La orden que ejecuta `system` tiene la siguiente forma:

```
ssh usuario@direccion [-p puerto] "qgen [opciones]" > qgen.txt
```

Se trata de una llamada a `ssh` a la que se le han añadido tres bloques de parámetros: En el primero (`usuario@direccion [-p puerto]`) se definen los parámetros que permiten la conexión a la máquina remota; esto es, el nombre de usuario, la dirección IP y el puerto. El valor del puerto, que es el 22 por defecto, no se incluye a menos que el usuario haya especificado uno distinto mediante la opción `-H` (véase el manual de uso, en el anexo C).

El segundo grupo de parámetros ("qgen [opciones]") contiene la orden que se ejecutará desde la máquina remota una vez completada la autenticación.
El último grupo de parámetros (> qgen.txt) es una redirección del flujo de salida estándar STDOUT al fichero qgen.txt. De esta forma, todos los mensajes emitidos por el generador de páginas web son redirigidos hacia ese archivo en lugar de mostrarse por pantalla.
Si no se ha realizado la configuración de SSH explicada en el anexo B.4, OpenSSH solicitará por la línea de comandos la autenticación del usuario mediante la introducción de su contraseña en el equipo remoto:

```
Bienvenido a qBench, versión 1.3.8.X
Carlos Quesada Granja, 2008
* Ejecutando qGen en el servidor mediante conexión SSH.
root@localhost's password:
```

Si la contraseña no se proporciona correctamente en tres intentos, el programa termina.

2.6.4. Función LeerFicheroDatos

Esta función se encarga de leer y almacenar en variables los datos contenidos en el fichero de texto qgen.txt, generado tras una correcta ejecución de la función anterior. El contenido del fichero qgen.txt depende de si las páginas web se han creado con éxito en el servidor o no. A partir del análisis de este fichero, la función decide si el programa se debe seguir ejecutando o si debe parar y mostrar un error.
Cada línea de texto del fichero qgen.txt representa uno de los ficheros generados. Las líneas se componen de varios elementos, que siguen un formato específico dependiendo de lo que representen. El número de elementos de cada línea será diferente si los archivos generados son estáticos o dinámicos, ficheros-objeto o páginas web. La función lee cada una de estas palabras y las guarda en variables (miembros de la estructura INPUT) para su uso posterior. Puesto que el número de variables que se tienen que guardar es desconocido en el tiempo de compilación del programa, éstas se tienen que crear de forma dinámica durante el tiempo de ejecución. La función ReservarMemoriaFicheros se encarga de ello antes de leer el fichero.
Por ejemplo, para una ejecución del benchmark en la que se generan cuatro ficheros-objeto y tres páginas estáticas, qgen.txt ofrecería el siguiente resultado:

```
/var/www/obj0001.html 0000000000000925
/var/www/obj0002.html 0000000000000119
```

```
/var/www/obj0003.html 0000000000000204
/var/www/obj0004.html 0000000000000238
/var/www/pag0001.html 0000000000000722 00000008
/var/www/pag0002.html 0000000000000645 00000009
/var/www/pag0003.html 0000000000000377 00000003
```

Cada línea que representa un fichero-objeto se compone de dos elementos, mientras que las que representan páginas web estáticas tienen tres. El primero de estos elementos (para cualquiera de los dos tipos de fichero) es una cadena de caracteres de tamaño variable que describe la ruta completa en la que se encuentra el fichero creado. El segundo de los elementos es una cifra de dieciséis dígitos en la que se guarda el tamaño en bytes de los ficheros creados. Por último, las páginas estáticas tienen un tercer bloque de ocho dígitos que indica el número de objetos que componen cada página.

Una ejecución del benchmark que tuviera que crear cinco páginas dinámicas ofrecería el siguiente resultado para `qgen.txt`:

```
/var/www/pag0001.php 0000000000003634 00000004 0000000000003099
/var/www/pag0002.php 0000000000005375 00000006 0000000000005168
/var/www/pag0003.php 0000000000003636 00000004 0000000000005708
/var/www/pag0004.php 0000000000007983 00000009 0000000000007217
/var/www/pag0005.php 0000000000002764 00000003 0000000000002729
```

Igual que en el caso anterior, los tres primeros elementos de cada línea se corresponden respectivamente con la ruta completa de la página, el tamaño en bytes de los ficheros y el número de objetos que debe mostrar cada página. El cuarto elemento, exclusivo de las páginas dinámicas, es una cifra de dieciséis dígitos que almacena la suma total de los tamaños en bytes de cada uno de los objetos de la página, es decir, el número total de caracteres mostrados.

La función se aprovecha de la uniformidad del formato del fichero `qgen.txt` para leer los valores mediante la función `fread`, de la biblioteca `stdio.h`. Si el fichero se encontrase vacío o no se pudiera leer correctamente (esto puede ocurrir porque el programa generador de páginas web no está instalado en la máquina servidora o porque ha habido un error en la ejecución de OpenSSH) se mostraría un mensaje de error (“`error al crear los ficheros`”) y se cerraría el programa.

Función ReservarMemoriaFicheros

Esta función se encarga de reservar memoria en tiempo de ejecución para guardar los datos del fichero `qgen.txt` en la estructura de datos `INPUT`. Esta estructura ya tiene preparada una serie de seis punteros (para guardar vectores de elementos) y dos punteros a puntero (para guardar vectores de cadenas de caracteres) con este fin. Sólo se reservará la memoria de aquellas variables necesarias, en función de si los archivos generados son dinámicos o estáticos.

2.6.5. Función ConectarMonitor

`ConectarMonitor` realiza una conexión SSH con el equipo servidor, similar a la realizada por la función `ConectarGeneradorFicheros`, para iniciar la ejecución del programa monitor de procesos. También hace uso de la instrucción `system`, aunque en este caso la orden emitida es menos compleja:

```
ssh usuario@direccion [-p puerto] "qmon [opciones]"
```

Esta orden se diferencia de la realizada por la función `ConectarGeneradorFicheros` porque el programa al que llama es diferente (a `qmon` en vez de a `qgen`) y porque no hay salidas que necesiten ser redireccionadas a ningún fichero. Además, hay que destacar que la llamada a la función `system` se realiza desde un hilo de ejecución diferente (usando la instrucción `fork`, de la biblioteca `unistd.h`). Si no se hiciera así, el programa principal quedaría bloqueado a la espera de la finalización de la ejecución del monitor (que no ocurriría nunca).
La función tiene en cuenta dos aspectos importantes: el primero es que, justo antes de realizar la conexión SSH, realiza una llamada a la función `ParарMonitor` (comentada más adelante, en la sección 2.6.8) para detener cualquier instancia del programa monitor que se pudiera estar ejecutando previamente en el servidor. El segundo aspecto está relacionado con el control de señales (función `signal` de la biblioteca `signal.h`). En el caso de que el usuario interrumpa la ejecución del programa principal mediante alguna combinación de teclas (como CTRL+C), el programa mostrará por pantalla el siguiente mensaje: "*AVISO: No ha sido posible detener qMon en el servidor; termine el proceso manualmente*".

2.6.6. Función ComprobarPeticiones

La función `ComprobarPeticiones` crea dos vectores aleatorios: uno con los valores de los tiempos en los que se deben realizar las peticiones HTTP al servidor, y otro con los

identificadores de las páginas web que se deben solicitar en cada petición.
Antes de generar cualquier número aleatorio, la función llama a `EstablecerSemilla`, que determina el modo en el que se generan los números aleatorios a partir de una semilla.
Para el vector de tiempos, se obtiene un vector de valores aleatorios mediante la función `GenerarVectorAleatorio`. Estos valores se generan en base a una distribución de probabilidad previamente seleccionada por el usuario. Una vez obtenido el vector, se procede a sumar el elemento i con su anterior $i-1$ (acumulación de valores sucesivos). De esta forma se obtiene un vector creciente en el que la diferencia de un elemento con su anterior (el intervalo de espera, ya que representan tiempos) tiene la distribución de probabilidad escogida por el usuario.
La obtención del vector con los identificadores de las páginas es más sencilla, ya que los valores se asignan al azar (distribución uniforme).

Función EstablecerSemilla

Ésta es una función sencilla que tiene como objetivo establecer la semilla de la función generadora de números aleatorios `drand48`, analizada en la sección 2.3.1. Si el usuario ha introducido por la línea de comandos un argumento para la opción `-s`, ése será el valor empleado. En caso contrario, se hará una llamada a la función `time`, que devolverá el número de segundos transcurridos desde el 1 de enero de 1970, y será el valor que se utilice como semilla.

Función GenerarVectorAleatorio

`GenerarVectorAleatorio` es utilizada tanto por el generador de carga como por el generador de páginas web. Se encarga de crear vectores de números aleatorios con una distribución de probabilidad previamente especificada por el usuario.
Las elementos que componen los vectores son del tipo `double`, pero las funciones que trabajan con ellas se encargan de adaptarlas a otros tipos según sus necesidades. Por ejemplo, cuando los valores se utilicen para definir tamaños de página (en bytes) se tratarán como tipo `long`, pero cuando se utilicen para definir el número de objetos por página se tratarán como tipo `int`. En algunas adaptaciones se pasa de datos en coma flotante a enteros, lo que implica truncar la parte decimal.
Existen seis funciones, integradas en una estructura de control `switch-case`, a las que `GenerarVectorAleatorio` puede llamar. Cada función devuelve un vector de cualquier tamaño cuyos elementos siguen la distribución de probabilidad que se indica:

1. `DistribucionU`: uniforme (sección 2.3.3).

2. `DistribucionE`: exponencial (sección 2.3.4).

3. `DistribucionH`: hiperexponencial (sección 2.3.5).

4. `DistribucionP`: de Pareto (sección 2.3.6).

5. `DistribucionL`: lognormal (sección 2.3.7).

6. `DistribucionC`: el vector generado contiene un valor constante en todos sus elementos. Es equivalente a una distribución uniforme en la que los valores mínimo y máximo coinciden.

2.6.7. Función CrearProcesos

Esta función es la más importante y compleja del programa. Es la encargada de enviar las peticiones HTTP al servidor y de recibir su respuesta. Esto implica que ha de llevar una correcta temporización de las peticiones para respetar la distribución de probabilidad establecida y que, además, tiene que medir el tiempo de respuesta de las peticiones y el tamaño de los paquetes recibidos para poder elaborar las estadísticas finales. En la figura 2.10 se muestra un diagrama de flujo simplificado de la función.
Para controlar el envío de las peticiones se ha recurrido a un diseño de múltiples hilos de ejecución. Si el programa se ejecutara en un único hilo de ejecución las peticiones y sus respuestas sólo se podrían atender de forma secuencial. Si en el intervalo de tiempo en el que se estuviera atendiendo una petición fuera necesario realizar una nueva petición, el programa no podría llevarla a cabo. La nueva petición tendría que esperar a que el programa terminase con la petición anterior para que pudiera empezar con ella. De esta forma, sería inevitable que la nueva petición sufriera un retraso con respecto a su tiempo original de inicio. Si cada petición tuviera que realizarse dentro de un intervalo de tiempo inferior al tiempo de respuesta, la ejecución del programa con un único hilo de ejecución sería inviable. Para que varias peticiones puedan ser enviadas de forma concurrente es necesario que el programa sea multihilo.
El programa se ha diseñado para que cada petición se realice en un proceso diferente. El método seguido para conseguirlo se comenta a continuación:

1. Se pone en marcha el reloj que se toma como referencia para controlar el inicio de las peticiones HTTP (función `ActivarCrono`). Véase la sección sobre las funciones-cronómetro más adelante.

2. El programa entra en un bucle `for` del que no sale hasta que no se hayan realizado todas las peticiones.

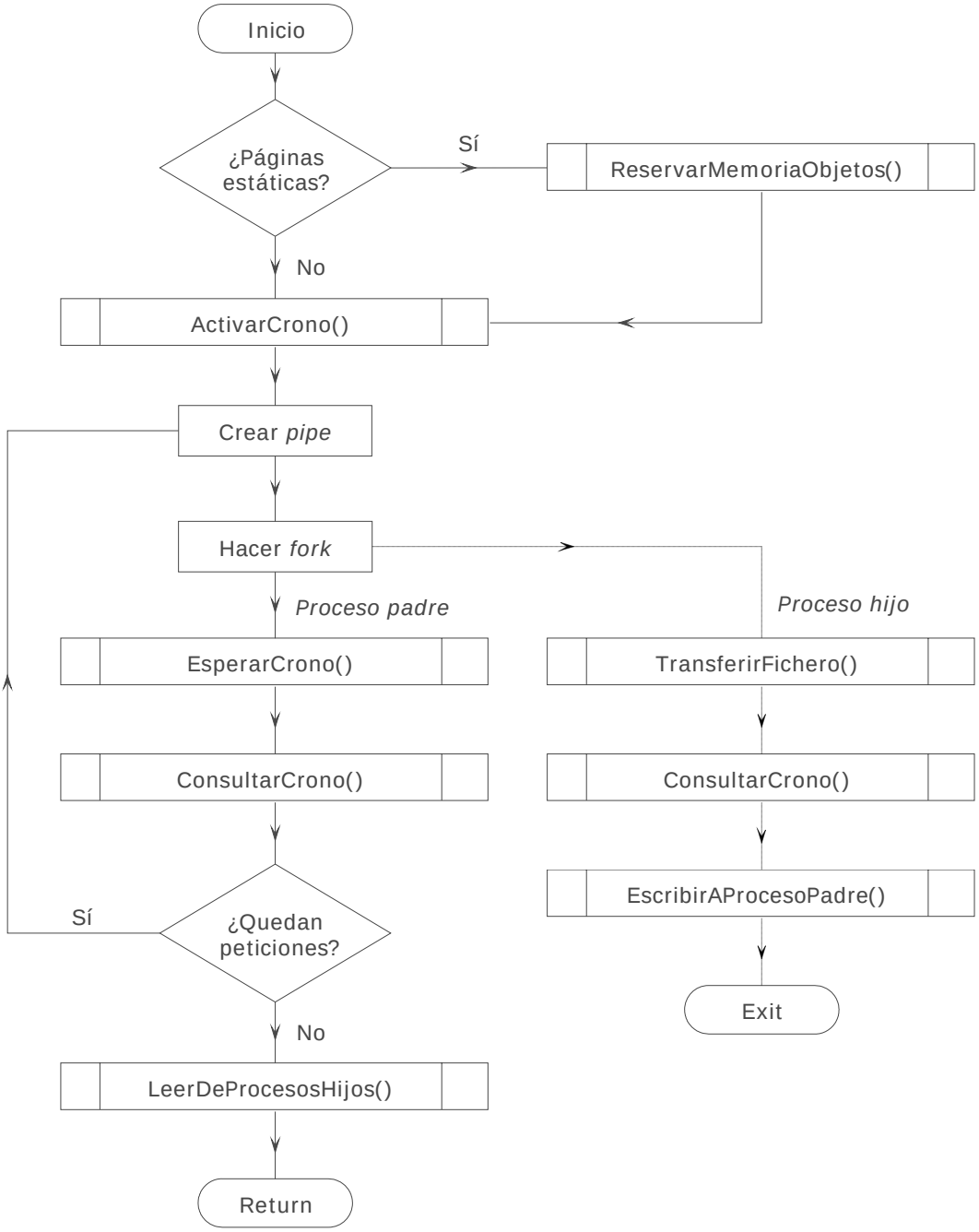

Figura 2.10: Diagrama de flujo simplificado de `CrearProcesos`.

3. Se crea un *pipe* —un objeto que permite el flujo de datos entre procesos—, que se usará para comunicar el futuro proceso hijo con el proceso padre. En el establecimiento de un *pipe* se emplean dos descriptores de fichero.

4. Mediante un *fork* se crea un proceso hijo independiente del padre. Cada proceso tiene una función bien definida:

a) El proceso hijo realiza de forma inmediata una petición HTTP al servidor (función `TransferirFichero`) y espera lo necesario hasta obtener su respuesta. Después, consulta el reloj para averiguar el tiempo que se ha invertido en el intercambio (función `ConsultarCrono`). Este dato y algunos más se escriben al padre (función `EscribirAProcesoPadre`) a través del *pipe* establecido anteriormente. A continuación, el proceso hijo finaliza.

b) El proceso padre espera una cantidad exacta de microsegundos hasta que sea el momento de enviar una nueva petición (funciones `EsperarCrono` y `ConsultarCrono`). Luego inicia un nuevo ciclo del bucle (vuelta al punto 3).

5. Cuando no quedan más peticiones por realizar, el proceso padre sale del bucle y lee los datos que le han transferido sus hijos por cada uno de los *pipes* (función `LeerDeProcesosHijos`).

En el esquema anterior se observa cómo la creación de nuevos procesos se va repartiendo en el tiempo según se vayan necesitando, de acuerdo con la distribución de probabilidad escogida por el usuario. Se ha comprobado experimentalmente que, si por el contrario, se crean a la vez todos los procesos en un primer momento y éstos tienen que esperar el instante adecuado para enviar la petición, el programa se satura.
Los descriptores de fichero son el factor limitante más importante de la comunicación entre procesos mediante *pipes*. Los descriptores de fichero son recursos limitados de los ordenadores, necesarios para transferir información entre procesos. Por cada *pipe* que se crea se asignan dos descriptores de fichero, y muchas distribuciones de Linux restringen su uso a un máximo de 1 024 por defecto.
Este programa se ha diseñado para reducir al mínimo el uso de descriptores de fichero, pues sólo emplea un único *pipe* por cada proceso hijo. No es posible establecer una comunicación entre el proceso padre y sus hijos con un mismo *pipe* porque todos los hijos escribirían sus datos en la misma posición de memoria; los datos que escribiera un proceso hijo serían remplazados por los datos del siguiente.
Con 1024 descriptores de fichero sólo se pueden crear 512 *pipes*. Es decir, sólo se pueden realizar 512 peticiones HTTP al servidor web. Para aumentar este límite, basta con hacer uso de la orden `ulimit` tal y como se explica en las recomendaciones del apéndice C (sección C.4).

Función ReservarMemoriaObjetos

Esta función es muy parecida a `ReservarMemoriaFicheros`, vista en la sección 2.6.4, pero sólo se ejecuta si las páginas web con las que trabaja el benchmark son estáticas.

Se encarga de reservar memoria en tiempo de ejecución para guardar los datos de los ficheros-objeto recibidos (los valores de los tiempos de respuesta y los tamaños) en la estructura de datos `OUTPUT` (comentada en la sección 2.5.1).

Las funciones-cronómetro: ActivarCrono y ConsultarCrono

Las peticiones HTTP que el programa envía al servidor web son para solicitar dos tipos de ficheros: páginas web (que pueden ser estáticas o dinámicas) y los ficheros-objeto (que componen las páginas web estáticas). Por cada petición que se realice se debe medir el tiempo de respuesta. Esto es, el tiempo transcurrido desde que se envía la petición al servidor hasta que se recibe el fichero completo.
Para medir estos intervalos de una forma sencilla, se obtienen los *timestamps* (secuencias de carácteres que denotan la fecha y el tiempo en el que ha ocurrido un evento) de los momentos en los que se inician y se finalizan las transferencias de los ficheros en base a un mismo reloj, y se resta el segundo valor con el primero.
El reloj con el que se toman dichos *timestamps* viene dado por la función `gettimeofday`, de la biblioteca `sys/time.h`. Esta función proporciona mediante la estructura `timeval` (definida en la misma biblioteca) el tiempo transcurrido en microsegundos desde la medianoche del 1 de enero de 1970. Es una función bastante similar a `time` (página 17) aunque tiene diferencias notables en la precisión y en el uso.
La función `ActivarCrono` se utiliza para comenzar a medir el tiempo (obtiene el primer *timestamp*). Sólo se usa una vez, dentro de `CrearProcesos`, justo antes de realizar las peticiones. La función llama a `gettimeofday` y guarda el resultado en una variable de la estructura `timeval`, redefinida bajo el nombre de `CHRONO` en el módulo `qlib` e integrada en la estructura `INPUT` (sección 2.5.1).
La función `ConsultarCrono` devuelve el número de microsegundos transcurridos desde que se llamó a la función `ActivarCrono`. Para conseguirlo, llama a la `gettimeofday` y resta el valor obtenido con el almacenado en la estructura `INPUT`.

Función EsperarCrono

La función `EsperarCrono` permite que el proceso padre de la función `CrearProcesos` espere un cierto número de microsegundos antes de iniciar un nuevo ciclo del bucle en el que se encuentra.
Existen al menos tres formas de conseguir que un conjunto de instrucciones se ejecute tras un intervalo de tiempo: mediante señales, mediante la función `nanosleep` y mediante comprobaciones sucesivas del reloj.

- **Mediante señales**. Las señales son una forma limitada de comunicación entre procesos usada en sistemas operativos tipo Unix, como Linux. Son notificaciones asíncronas que se envían a un proceso para comunicarle que ha ocurrido un evento. Cuando se envía una señal a un proceso, el sistema operativo interrumpe la ejecución normal de éste. Si el proceso había registrado previamente un procedimiento (*handler*), se ejecuta éste; si no, se ejecuta el procedimiento que la señal tenía por defecto. Existen funciones, como `alarm` de la biblioteca `unistd.h` o `setitimer` de `sys/time.h` que permiten generar señales SIGALRM tras un intervalo de tiempo.

  Uno de los problemas que tiene trabajar con señales es que los retrasos en la planificación del procesador pueden impedir que el proceso ejecute el procedimiento tan pronto como se genere la señal, lo que equivale a retrasos. Además, se ha comprobado experimentalmente que durante la espera el porcentaje de uso de la CPU del ordenador tiende a aumentar notablemente.

- **Mediante la función** `nanosleep`. Esta función de la librería `time.h` permite suspender la ejecución de un proceso durante un intervalo de tiempo medido en nanosegundos. Existen funciones derivadas, como `usleep` y `sleep`, que permiten medir el tiempo en microsegundos y en segundos, respectivamente. La ventaja de esta función es que el proceso no consume recursos de la CPU mientras está suspendido.

  Las desventajas son, por un lado, que el tiempo en el que el proceso se encuentra suspendido (o dormido) puede ser mayor de lo especificado debido a latencias del sistema y posibles limitaciones en la resolución del temporizador del *hardware*. Y por otro lado, que algunos tipos de señales pueden hacer que el proceso "se despierte" anticipadamente.

- **Mediante comprobaciones sucesivas del reloj**. Consiste en introducir el proceso dentro de un bucle de lectura del reloj del sistema (usando la función `gettimeofday`) del que sólo puede salir cuando el valor del tiempo leído es superior al valor del tiempo puesto como límite. El problema de esta técnica es que los bucles que se prolongan durante muchos ciclos (del orden de algunos segundos) suelen llevar al procesador al máximo de su capacidad.

Para abordar el problema de la forma menos compleja y haciendo el menor uso del procesador posible, se ha optado por una solución mixta entre los dos últimos puntos. La función `EsperarCrono` utiliza `usleep` para esperar hasta los 15 milisegundos anteriores al término del intervalo. En ese momento la forma de esperar cambia a compro-

baciones sucesivas del reloj. Al permitir que los últimos 15 milisegundos de la espera sean comprobados de una manera más minuciosa se está asegurando que la exactitud del tiempo medido es mayor que cuando sólo se confía con la función `usleep`. Y puesto que el tiempo en el que se realizan las comprobaciones sucesivas es muy breve, el uso de la CPU permanece bajo.

No obstante, debido al comportamiento imprevisible del procesador, pueden surgir algunos retrasos del orden de milisegundos que, en principio, no tienen por qué alterar demasiado la distribución de probabilidad que modela los tiempos de espera. Esta cuestión se analiza con profundidad en el capítulo 4.

Función TransferirFichero

Es la función que se utiliza para obtener los ficheros del servidor web. En la figura 2.11 se muestra su diagrama de flujo. Primero crea y conecta un *socket* (función `CrearYConectarSocket`), después envía la petición HTTP al servidor (función `EnviarPeticionHTTP`), y cuando la recibe (función `RecibirRespuestaHTTP`) cierra el *socket*. Esta secuencia responde al esquema de conexiones múltiples comentado en la sección 2.2.2.

Si las páginas son estáticas, la función utiliza recursividad (se llama a sí misma) para descargar cada uno de los objetos. En ese caso, también controlará sus tiempos de respuesta. En las secciones siguientes se comentan las funciones que componen `TransferirFichero`.

Función CrearYConectarSocket

`CrearYConectarSocket` llama adecuadamente a las funciones `socket` y `connect`, de las bibliotecas `sys/types.h` y `sys/socket.h` (se requieren ambas para cada función), y son el primer paso para el establecimiento de la comunicación con el servidor. En la sección 2.2.4 se detalló el funcionamiento de los *sockets*.

Función EnviarPeticionHTTP

La función `EnviarPeticionHTTP` crea una cadena de caracteres con una petición HTTP como la especificada en la sección 2.2.2, con la que solicita la transferencia de una página web o de un fichero-objeto (dependiendo del caso). El envío se realiza con la instrucción `send`, de la biblioteca `sys/socket.h`.

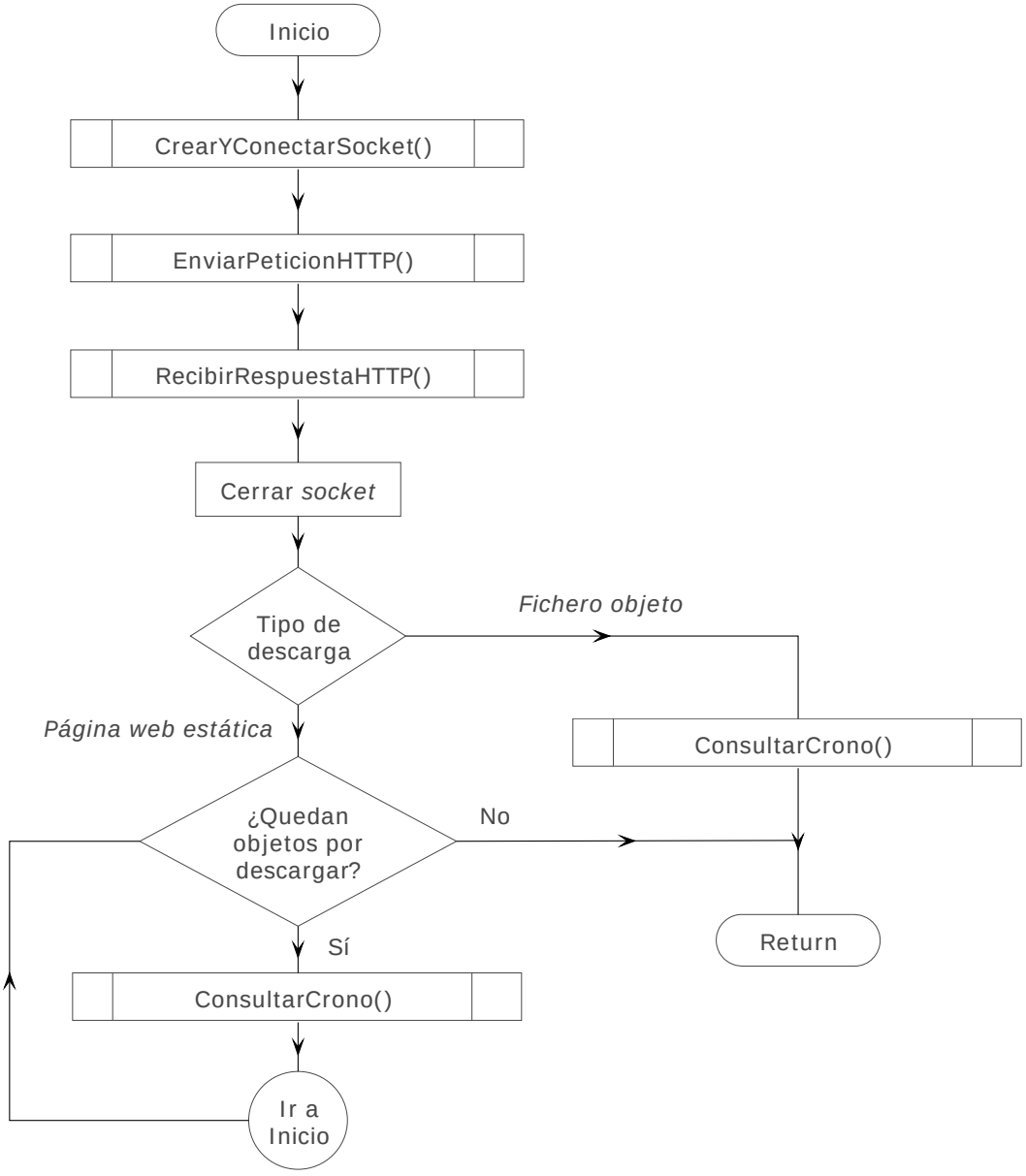

Figura 2.11: Diagrama de flujo simplificado de `TransferirFichero`.

Función RecibirRespuestaHTTP

Esta función utiliza la instrucción `recv`, de la biblioteca `sys/socket.h`, para recibir datos del *socket*. Por cada llamada a la función se lee un número constante de bytes, que pueden contener información sobre las cabeceras HTTP o sobre el contenido del fichero descargado.

No es un objetivo de este programa interpretar el significado de las cabeceras recibidas. Sólo es importante contar el número total de bytes recibidos para poder realizar un análisis posterior. Las llamadas a `recv` se realizan dentro de un bucle del que úni-

camente se sale cuando el número de bytes recibidos es cero, es decir, cuando se ha completado la transferencia.
Cuando las páginas descargadas son estáticas se hace necesario interpretar *al vuelo* el contenido de los datos recibidos, ya que se deben localizar los nombres de los ficheros-objeto para poderlos descargar. La función `AnalizarCadenas` tiene un algoritmo que permite conseguir esto.

Función AnalizarCadenas

`AnalizarCadenas` puede localizar en una cadena otras subcadenas que tengan como prefijo los caracteres "`src="`" y como sufijo un carácter de punto ".".
Las subcadenas que cumplen esos requisitos contienen parte del nombre de los ficheros-objeto que conforman las páginas web estáticas. Cuando se localizan, se obtienen los cuatro dígitos que identifican a los ficheros y se almacenan en un vector.
Por ejemplo, en una cadena de texto que incluya una etiqueta HTML como la siguiente,

```
<frame src="obj0123.html">
```

que se utiliza para indicar la URL de un documento que debe ir dentro de un *frame*, la función `AnalizarCadenas` reconocerá la subcadena

```
src="obj0123.
```

ya que tiene "`src="`" como prefijo y "`.`" como sufijo.
De esta cadena, la función guardará en un vector el identificador "`0123`", que servirá para descargar más adelante el fichero-objeto al que hace referencia.
El programa puede detectar las subcadenas incluso cuando han sido divididas en dos partes por dos cadenas consecutivas. Esto es posible porque las variables de la función se han declarado como estáticas (`static`), lo que permite que mantengan sus valores entre llamadas.

2.6.8. Función PararMonitor

Esta función es muy similar a `ConectarMonitor` (sección 2.6.5). Su objetivo es detener la ejecución del programa monitor de procesos. Para ello, primero realiza una conexión SSH con el equipo servidor en la que emite la siguiente orden:

```
ssh usuario@direccion "ps -A | grep qmon | head -c 5" > .temp
```

`ps` es un programa que, acompañado de la opción `-A`, muestra todos los procesos del sistema. La salida de este programa se convierte en la entrada de `grep`, una utilidad que permite seleccionar sólo aquellas líneas de texto en las que aparezca la palabra `qmon`, es decir, el nombre del programa monitor de procesos. A continuación, `head -c 5` permite seleccionar los cinco primeros bytes (o caracteres) de las líneas filtradas, que se corresponden con los identificadores numéricos de los procesos (PID). Es decir, al final de la secuencia de *pipelines* se obtiene una lista con todos los PID del programa monitor de procesos. Esta lista se redirige al fichero oculto `.temp`.
El siguiente paso que realiza la función es abrir el fichero `.temp` y leer la primera línea. Si es un número, vuelve a hacer una conexión SSH con el servidor para matar el proceso al que identifica:

```
ssh usuario@direccion "kill XXXXX"
```

donde `XXXXX` es un PID. En este punto, la función se comporta de manera recursiva, y se llama a sí misma las veces necesarias hasta matar todos los procesos `qmon` activos del servidor. Normalmente habrá uno o ninguno, pero es importante comprobar que no hay más para no sobrecargar el procesador del servidor con procesos innecesarios.

2.6.9. Función MostrarResultados

Llama a las funciones que generan los ficheros con los resultados de la ejecución del programa (cuyos nombres por defecto son `result.csv` y `result.txt`) y muestra un mensaje por pantalla que le indica al usuario la finalización del mismo.

Función GenerarFicheroCSV

Los ficheros CSV son archivos de texto en los que se guardan series de datos separados por comas. En los ficheros CSV generados por este benchmark, cada línea de texto contiene los datos de una variable diferente, que se identifica por su nombre, que ocupa el primer elemento de cada fila. Las variables que se guardan se listan a continuación:

1. `fecha`. Contiene un *timestamp* en el que sus elementos están separados por espacios y no por comas. El formato que sigue es el siguiente: primero el día de la semana, seguido del nombre del mes, el día, la hora y el año. Por ejemplo: `Mon Mar 3 12:52:02 2008`.

2. `elementos`. Se compone de tres valores, que recogen los primeros atributos de las opciones obligatorias `-n`, `-o` y `-p` respectivamente. Por ejemplo, si las opciones que

se han pasado al programa son `-d -n 2,c,0.5 -o u,200,4000 -p 10,u,1,12`, los valores de esta variable serían `2`, `0` y `10`.

3. `distrProb`. Recoge los atributos de las opciones obligatorias `-n`, `-o` y `-p` que ocupan la segunda posición. Tomando el ejemplo anterior, los valores de la variable serían `c`, `u` y `u`.

4. `valorFDP[0]`. Recoge los atributos de las opciones obligatorias `-n`, `-o` y `-p` que ocupan la tercera posición. Con el ejemplo visto en el punto 2, los valores de la variable serían `500000`, `0` y `0`. No hay que olvidar que los valores en segundos se traducen internamente a microsegundos.

5. `valorFDP[1]`. Recoge los atributos de las opciones obligatorias `-n`, `-o` y `-p` que ocupan la cuarta posición. Con el ejemplo visto en el punto 2, los valores de la variable serían `200`, `4000` y `0`.

6. `valorFDP[2]`. Recoge los atributos de las opciones obligatorias `-n`, `-o` y `-p` que ocupan la quinta posición. Con el ejemplo visto en el punto 2, los valores de la variable serían `1`, `12` y `0`.

7. `tamanoPaginas`. Indica el tamaño en bytes de las páginas web que se han creado.

8. `objetosPagina`. Indica el número de objetos del que se compone cada una de las páginas web creadas.

9. `tamanoObjetosPagina`. Indica la suma de los tamaños en bytes de los ficheros-objeto que componen las páginas web que se han creado. El valor coincide con el número de caracteres de cada página.

10. `tiemposInicioPeticion`. Contiene los tiempos en microsegundos en los que teóricamente se debía iniciar cada petición al servidor.

11. `paginaPeticion`. Identifica las páginas web que se debían descargar en cada petición. Por ejemplo, si el primer elemento de la variable vale `4`, significa que la primera página que se ha solicitado ha sido la `pag0004.html` (o `.php`).

12. `tiempoInicioPag`. Contiene los tiempos en microsegundos en los que realmente se ha iniciado cada petición al servidor. Estos valores serán idénticos o ligeramente superiores a los contenidos en `tiemposInicioPeticion`.

13. `tiempoTransferPag`. Contiene los tiempos en microsegundos en los que se ha completado la transferencia de la página web.

14. `tamanoPaginasCC`. Guarda el tamaño en bytes de los paquetes recibidos durante la transferencia de cada página web, incluyendo las cabeceras del protocolo HTTP.

Si las páginas son estáticas, también se guardan variables relacionadas con los ficheros-objeto:

15. `numeroObj[n]`. Identifica los ficheros-objeto que ha solicitado la página web `n`. Por ejemplo, si el primer elemento de la variable `numeroObj[7]` vale `4`, significa que el primer objeto de la séptima página web solicitada es el fichero `obj0004.html`.

16. `tiempoInicioObj[n]`. Contiene los tiempos en microsegundos en los que se han iniciado las transferencias de los objetos de la página web `n`.

17. `tiempoTransferObj[n]`. Contiene los tiempos en microsegundos en los que se han completado las transferencias de los objetos de la página web `n`.

18. `tamanoObjetosCC[n]`. Guarda el tamaño en bytes de los paquetes recibidos durante la transferencia de los objetos que componen la página web `n`, incluyendo las cabeceras de las tramas HTTP.

La organización de estos datos en series de números separados por comas permite su tratamiento posterior con herramientas matemáticas como Matlab o Excel. En la sección 3.2 se comentan los programas que se han utilizado para analizar los resultados.

Función GenerarFicheroTXT

La función crea un fichero de texto en el que se muestran los datos más importantes obtenidos tras la ejecución del programa. Estos datos se ordenan por columnas, y pretenden servir como ayuda visual al usuario. Los datos mostrados son diferentes dependiendo de si las páginas que se descargan son estáticas o dinámicas.
Un ejemplo de fichero real con páginas estáticas es el siguiente:

Pag	N.Obj	Tam.Tot [B]	Tam.Recib [B]	T.Petic [s]	T.Transf [ms]
1	4	1419	2791	0.000	28.368
	1	236	510	0.009	4.319
	2	236	510	0.018	4.335
	3	265	540	0.022	3.080
	4	236	510	0.026	2.755

```
-------------------------------------------------------------------------
  2       4           1709          3083          0.500          7.317
-------------------------------------------------------------------------
          1            361           636          0.505          0.588
          2            236           510          0.505          0.596
          3            361           636          0.506          0.605
          4            305           580          0.507          0.721
-------------------------------------------------------------------------
  3       2            918          1737          1.000          3.237
-------------------------------------------------------------------------
          1            305           578          1.002          0.548
          2            305           578          1.003          0.412
-------------------------------------------------------------------------
  4       3           1370          2462          1.500          4.193
-------------------------------------------------------------------------
          1            305           578          1.503          0.466
          2            383           656          1.503          0.589
          3            305           578          1.504          0.596
[...]
```

Se comprueba de un solo vistazo que se han realizado cuatro peticiones al servidor y que las páginas descargadas en cada una de estas peticiones contenían 4, 4, 2 y 3 objetos respectivamente.

La columna etiquetada como `Tam.Tot [B]` indica el tamaño neto de los ficheros descargados; en un caso con la suma de las páginas y los objetos, y en otros casos desglosado para cada uno de los objetos. En `Tam.Recib [B]` se muestra la misma información que en la columna anterior, pero incluyendo también las cabeceras de las peticiones HTTP. La cuarta columna, `T.Petic [s]`, indica el tiempo en segundos en el que comenzó la descarga del fichero, mientras que la última, `T.Transf [ms]`, indica el tiempo de respuesta en milisegundos empleado en la descarga.

Si se observa la penúltima columna del ejemplo anterior, se comprueba cómo el tiempo en el que se realizan las peticiones de los objetos sigue un orden creciente (hay $0,5$ segundos de diferencia entre peticiones a páginas web). Esto es así porque en este caso había tiempo suficiente entre peticiones.

En el siguiente ejemplo real se ha reducido el tiempo entre peticiones para comprobar el funcionamiento multihilo del benchmark:

```
  Pag   N.Obj    Tam.Tot [B]  Tam.Recib [B]    T.Petic [s]  T.Transf [ms]
[...]
-------------------------------------------------------------------------
   2       3           1465           2564          0.002         44.456
-------------------------------------------------------------------------
           1            226            500          0.042          2.951
           2            487            762          0.045          0.631
           3            375            650          0.046          0.531
-------------------------------------------------------------------------
   3       3           1226           2326          0.004         74.737
-------------------------------------------------------------------------
           1            283            558          0.077          0.432
           2            283            558          0.077          0.557
           3            283            558          0.078          0.411
-------------------------------------------------------------------------
   4       3           1163           2262          0.006         74.306
-------------------------------------------------------------------------
           1            283            558          0.052          0.608
           2            283            558          0.053          0.658
           3            220            494          0.054         25.474
[...]
```

Se trata de una prueba en la que se ha iniciado una petición cada dos milisegundos. Se puede comprobar, viendo los valores de la penúltima columna, cómo el procesador consigue que esto se cumpla correctamente. Sin embargo, no tiene tiempo suficiente para descargar todos los objetos de cada página en 2 ms. Éstos se descargarán cuando la planificación del procesador lo permita, y no necesariamente en el orden que se realizaron las peticiones. Se puede ver cómo los objetos de la cuarta petición comenzaron su transferencia antes que los de la tercera petición.

Los ficheros generados cuando se trabaja con páginas dinámicas son algo diferentes:

```
  Pag   N.Obj    Tam.Tot [B]  Tam.Recib [B]    T.Petic [s]  T.Transf [ms]
-------------------------------------------------------------------------
   1       7           6243           2779          0.000        307.356
-------------------------------------------------------------------------
   2       8           7113           2734          0.500         20.346
-------------------------------------------------------------------------
   3       6           5373           2520          1.000         64.129
```

```
-------------------------------------------------------------------------
   4       5           4503           1689          1.500         19.158
-------------------------------------------------------------------------
   5       6           5373           2520          2.000         20.937
-------------------------------------------------------------------------
   6       8           7113           3138          2.500         24.901
[...]
```

El formato es similar al anterior, salvo por el hecho de que en las páginas dinámicas no hay ficheros-objeto para descargar. La columna `Tam.Tot [B]` muestra en este caso el tamaño de la página HTML, que integra un fragmento de código PHP por cada objeto. El tamaño recibido es el de las páginas ya procesadas por el servidor más las cabeceras de las peticiones HTTP.

2.7. El generador de páginas web

La función `main` del programa generador de páginas web (`qgen.c`), cuya sintaxis puede encontrarse en el anexo C.2, está compuesta por las seis líneas de código que se muestran a continuación:

```
int main(int argc, char **argv) {
     INPUT pm;
     if (LeerOpciones(argc, argv, &pm, qGENER)) return -1;
     if (ComprobarOpciones(&pm, qGENER)) return -1;
     if (ProcesarOpcionExcluyente(pm)) return -1;
     return 0; }
```

De las tres funciones que integran el programa, las dos primeras ya han sido comentadas en las secciones 2.6.1 y 2.6.2 respectivamente. La función restante, y todas las que se encuentran incluidas en ella, se detallan a continuación.

2.7.1. Función ProcesarOpcionExcluyente

La función `ProcesarOpcionExcluyente` se encarga de seleccionar unas funciones u otras dependiendo de si las páginas web que se tienen que generar son estáticas o dinámicas. En la figura 2.12 se muestra un diagrama de flujo simplificado de la función. En el caso de las páginas estáticas, la función llamará a `GenerarFicherosObjeto` para crear los ficheros-objeto y a `GenerarPaginasEstaticas` para crear los ficheros con

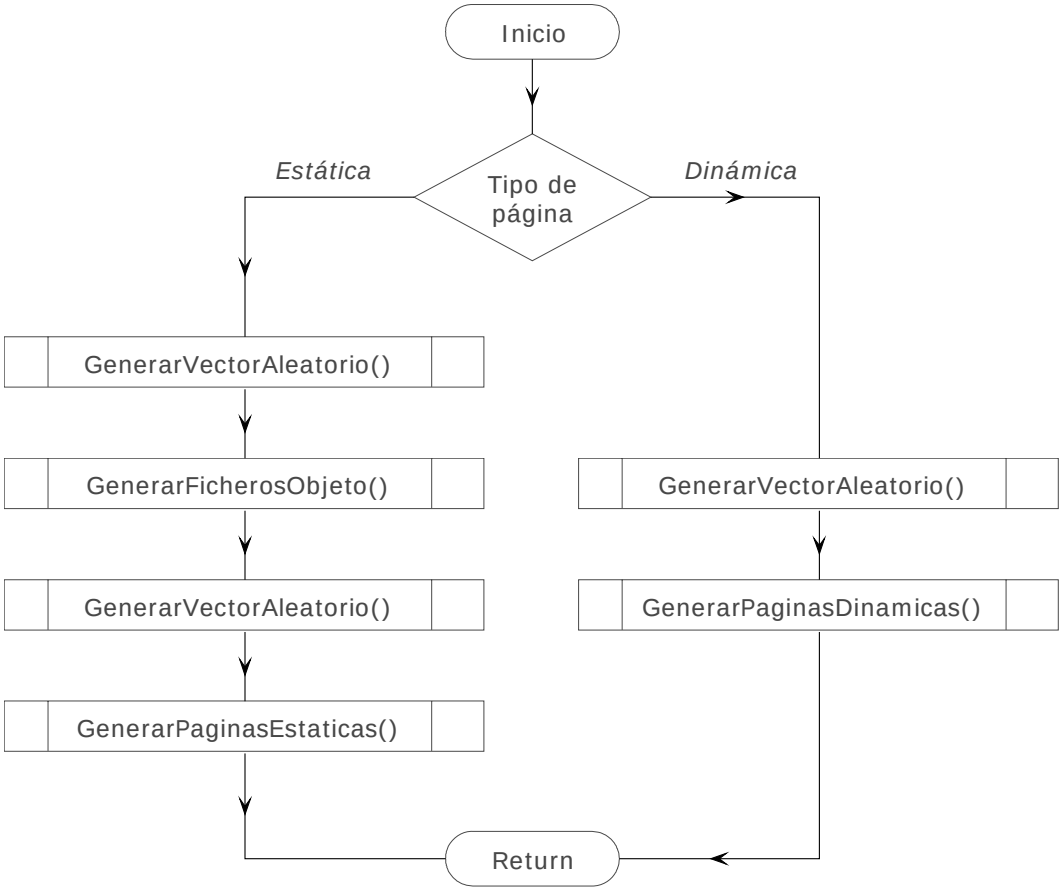

Figura 2.12: Diagrama de flujo simplificado de `ProcesarOpcionExcluyente`.

el código HTML. Cuando las páginas sean dinámicas, la función creará los archivos mediante `GenerarPaginasDinamicas`. Para cualquiera de las tres funciones anteriores es necesario llamar previamente a `GenerarVectorAleatorio`. Esta función, que ya se comentó en la sección 2.6.6, se encarga de obtener un vector de valores aleatorios que se usa en la creación de los ficheros.

Función GenerarPaginasEstaticas

Ésta es la función encargada de crear las páginas web estáticas que se comentaron en la sección 2.4.1. Para ello, la función usa la instrucción `fwrite`, de la biblioteca `stdio.h`, que permite escribir datos en un fichero y, al mismo tiempo, almacenar en una variable el tamaño de lo escrito. La función tiene definidas como constantes de cadenas de caracteres las etiquetas HTML que necesita, y las va escribiendo en los ficheros de acuerdo a la distribución por filas y columnas que proporcionan las ecuaciones vistas en 2.29. Para dar nombre a los ficheros, llama a la función `NombrarFichero`.

Función GenerarFicherosObjeto

Los ficheros-objeto son ficheros de texto con extensión `html` que simulan elementos HTML (marcos, tablas, listas), imágenes, archivos empotrados, etc. de una página web. La función `GenerarFicherosObjeto` se encarga de crear los ficheros-objeto a partir de un vector, que indica el tamaño que debe tener cada uno.
El modo en el que se generan los ficheros se explicó en la sección 2.4.1: se toman caracteres al azar del siguiente vector

```
"aaaaaaaaabccccdddddeeeeeeeeeeefghiiiiijkllllmmmnnnnnooooooooppqrrrrrsss
sssttttuuuvwxyz                "
```

hasta que se completa el tamaño requerido. El vector consta de 84 letras y 16 espacios en blanco. Se ha intentado que las letras tengan una distribución similar a la del idioma español para que haya más probabilidades de que se formen palabras con significado.
El nombre de cada fichero viene determinado por la función `NombrarFichero`.

Función GenerarPaginasDinamicas

La función `GenerarPaginasDinamicas` se encarga de crear páginas web dinámicas como las comentadas en la sección 2.4.2. Es muy similar a `GenerarPaginasEstáticas`; se diferencia de ésta en las constantes de cadenas de caracteres (que contienen largas secuencias de instrucciones PHP), y en el modo en que se distribuyen los objetos (uno tras otro). Como en las anteriores funciones, los nombres de los ficheros son determinados por la función `NombrarFichero`.

Función NombrarFichero

La función `NombrarFichero`, que se llama desde las tres funciones generadoras de ficheros anteriores, devuelve cadenas de caracteres con los nombres que llevarán los ficheros-objetos y las páginas web. El formato de estos nombres es el siguiente:

- Los primeros caracteres son secuencias de tres letras que indican si el fichero se trata de un objeto ("`obj`") o una página web ("`pag`").
- Los cuatro caracteres siguientes son dígitos, e identifican secuencialmente a cada tipo de fichero ("`0001`", "`0002`", "`0003`", etc.).
- Los últimos caracteres se emplean para especificar la extensión del fichero. Ésta será "`.html`" para las páginas estáticas o "`.php`" para las dinámicas.

2.8. El monitor de procesos

El programa monitor de procesos (`qmon.c`), cuya sintaxis puede encontrarse en el anexo C.3, tiene el siguiente código en su función principal:

```
int main(int argc, char **argv) {
    INPUT pm;
    if (LeerOpciones(argc, argv, &pm, qMONIT)) return -1;
    if (ComprobarOpciones(&pm, qMONIT)) return -1;
    Monitorizar(pm);
    return 0; }
```

Se compone de seis líneas y sólo llama a tres funciones, de las cuales `LeerOpciones` y `ComprobarOpciones` ya fueron comentadas en las secciones 2.6.1 y 2.6.2, respectivamente. La otra función, `Monitorizar`, se analiza a continuación.

2.8.1. Función Monitorizar

Esta función realiza un seguimiento, mientras se ejecuta el benchmark, del uso que hacen del procesador del equipo servidor las aplicaciones implicadas en la entrega de páginas web. Si las páginas web solicitadas son estáticas sólo se monitorizan los procesos etiquetados como `apache`, mientras que si son dinámicas también se monitorizan los etiquetados como `mysql`. De esta forma, se puede conocer el *esfuerzo* al que se ven sometidas las aplicaciones Apache y MySQL durante la ejecución del benchmark.
Para llevar a cabo el seguimiento, la función crea un fichero por cada proceso monitorizado. Por defecto, reciben los nombres de `result.qmon.apa.txt` (para Apache) y `result.qmon.mys.txt` (para MySQL). Luego, la función entra en un bucle infinito con un ciclo cada segundo. En cada ciclo, se añaden a los ficheros varias líneas con un *timestamp* e información sobre los procesos implicados, a modo de *log*. Para añadir la fecha a los ficheros, se hace una llamada a la instrucción `system` (comentada en la sección 2.6.3, página 42) para que ejecute la siguiente orden:

```
date "+%d/%m/%Y %H:%M:%S" >> result.qmon.XXX.txt
```

Con esta orden se le está pidiendo a la línea de comandos que escriba en el fichero `result.qmon.XXX.txt` (donde `XXX` depende del proceso que se esté monitorizando) un *timestamp* con el siguiente formato: `24/11/2007 14:30:00`.
Para añadir la información sobre los procesos en los ficheros, se hace otra llamada a la

instrucción `system` pero con una de estas dos órdenes (dependiendo del proceso):

```
ps aux | grep apache >> result.qmon.apa.txt
ps aux | grep mysql  >> result.qmon.mys.txt
```

Aquí se muestra un ejemplo de los datos que se pueden obtener tras la ejecución del monitor durante dos segundos:

```
USER       PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
25/03/2008 07:33:22
root      5341  0.4  1.2  22776  6572 ?   SNs  Apr23 0:00 /usr/sbin/apac...
www-data  5347  0.6  1.1  23044  5792 ?   SN   Apr23 0:00 /usr/sbin/apac...
root      9123  0.0  0.0   1712   476 ?   S    07:33 0:00 sh -c ?ps aux ...
root      9125  0.0  0.1   2900   768 ?   R    07:33 0:00 grep apache
25/03/2008 07:33:23
root      5341  0.4  1.2  22776  6572 ?   SNs  Apr23 0:00 /usr/sbin/apac...
www-data  5347  0.7  1.1  23044  5792 ?   SN   Apr23 0:00 /usr/sbin/apac...
root      9135  0.0  0.0   1712   480 ?   S    07:33 0:00 sh -c ?ps aux ...
root      9137  0.0  0.1   2900   772 ?   R    07:33 0:00 grep apache
```

Los datos de la tercera columna (`%CPU`) indican el porcentaje del procesador consumido por cada proceso. Para analizar estos ficheros de forma adecuada, se ha realizado un pequeño programa en Matlab que representa en una gráfica el consumo total del procesador en cada momento de la ejecución (véase la sección 3.2).
Puesto que el programa se ejecuta dentro de un bucle sin salida, para detenerlo es necesario interrumpirlo de alguna forma. Si el monitor se ejecuta como parte del benchmark, de ello se encarga la función `ParарMonitor`, vista en la sección 2.6.8. Si se ejecuta de forma manual, es necesario pulsar CTRL + C o una combinación similar.

Capítulo 3

RESULTADOS

En el presente capítulo el benchmark se somete a un gran número de ejecuciones de diferentes características y sobre diferentes topologías de red. Los ficheros obtenidos tras las ejecuciones se usarán más adelante para validar el funcionamiento del benchmark. En la sección 3.1 se explica el procedimiento que se ha seguido para realizar estas ejecuciones. Debido al gran número de ficheros de resultados obtenidos, se han desarrollado dos programas en Matlab para poder analizarlos, que se explican en la sección 3.2. En las secciones 3.3, 3.4 y 3.5 se exponen gráficas y tablas elaboradas a partir de los resultados, y en la sección 3.6 se extraen algunas conclusiones generales.

3.1. Realización de las pruebas

Para comprobar el correcto funcionamiento del benchmark se han diseñado nueve pruebas, cada una de ellas consistente en diez ejecuciones del programa, de diez minutos de duración cada una.
En cada prueba, los parámetros que definen el tiempo entre peticiones (`-n`), el número de objetos por página (`-p`) y el tamaño de los objetos (`-o`) se caracterizan mediante distribuciones de probabilidad diferentes (los valores de los atributos que acompañan a cada distribución se especifican más adelante, en los cuadros 3.1 y 3.39). Estas nueve pruebas se han realizado sobre tres topologías de red diferentes (localhost, red local e Internet) y con los dos tipos de páginas web disponibles (estáticas y dinámicas).
En total se han realizado 540 ejecuciones del benchmark, que equivalen a 90 horas de ejecución continuada, para los seis casos siguientes:

1. páginas estáticas en localhost (sección 3.3.1);

2. páginas dinámicas en localhost (sección 3.3.2);
3. páginas estáticas en red local (sección 3.4.1);
4. páginas dinámicas en red local (sección 3.4.2);
5. páginas estáticas en Internet (sección 3.5.1);
6. páginas dinámicas en Internet (sección 3.5.2).

Para automatizar lo máximo posible este trabajo se ha desarrollado un *script* en Perl, llamado `qbucle2`, que realiza las diez ejecuciones de las nueve pruebas de forma secuencial, de modo que, para una misma prueba, cada una de las diez ejecuciones diferentes que la componen se realiza con más de una hora y media de diferencia. En el código de `qbucle2` se pueden ver las 54 combinaciones de parámetros que se le han pasado al benchmark ordenadas por categorías.
Tras la realización de las pruebas se obtienen varios ficheros `txt` y `csv`. Debido a la gran cantidad de pruebas realizadas, el conjunto de los archivos generados ocupa más de 9 gigabytes de espacio (todos los ficheros se han en el DVD adjunto; véase el anexo A). Las ejecuciones son totalmente reproducibles, ya que el generador de números pseudoaleatorios se ha establecido con una semilla de valor $s = 241181$ para todas las pruebas.

3.2. Programas para el análisis de resultados

Como ya se vio en 2.6.9, los ficheros `txt` y `csv` obtenidos tras la ejecución del programa contienen datos relevantes sobre los tiempos de descarga de las páginas web y sus objetos, los tamaños de los ficheros recibidos y los porcentajes de uso del procesador, entre otros.
Estos datos se guardan como largas cadenas de números separadas por comas (en el caso de los ficheros `csv`) o como líneas de texto en las que la información se encuentra un poco más dispersa (en el caso de los ficheros `txt`). En ambos casos, los datos requieren un procesamiento previo, mediante algún programa de análisis numérico, para poder obtener resultados concretos.
En este proyecto se ha utilizado Matlab, un entorno que facilita la manipulación de matrices, la representación gráfica de datos y la implementación de algoritmos, mediante un lenguaje de programación propio conocido como código M.
Se han desarrollado dos programas en código M para analizar los resultados de las ejecuciones: uno para elaborar gráficas (`qgraf2`) y otro para elaborar tablas de datos

estadísticos (`qestad`). Estos programas se han empleado para obtener las gráficas y las tablas que se muestran en las secciones 3.3, 3.4 y 3.5. Las características de ambos programas se comentan a continuación.

3.2.1. El programa para elaborar gráficas

Este programa (`qgraf2`) se encarga de representar dos tipos de gráficas: por un lado, los tiempos de respuesta de las páginas web solicitadas en cada prueba; y por otro lado, el porcentaje del consumo medio del procesador que el servidor ha necesitado durante la ejecución del benchmark.

Para realizar las representaciones, el programa necesita los conjuntos de ficheros `*.csv`, `*.qmon.apa.txt` y `*.qmon.mys.txt`, obtenidos de las diez ejecuciones de una misma prueba.

Todas las gráficas se representan mediante el 95 percentil. Con ello se está indicando que, para un 95 % de las ejecuciones realizadas, las variables se encuentran por debajo del valor indicado, mientras que para el 5 % de las ejecuciones lo sobrepasa.

El cálculo de percentiles suele ser apropiado para la planificación y gestión de redes, ya que permite medir el tráfico cursado por el cliente descartando (o admitiendo) los picos de tráfico generados.

Para calcular el 95 percentil de las variables se utiliza la instrucción `prctile`, y para representar las gráficas resultantes en una misma ventana se emplean las instrucciones `plot` y `subplot`.

Obtención del tiempo de respuesta

Para obtener los vectores con los tiempos de inicio y finalización de transferencia de cada fichero `csv` resultante de la ejecución, el programa utiliza la instrucción de Matlab `csvread`. Al restar el segundo vector con el primero se obtiene el vector de tiempos de respuesta de las páginas.

En cada ejecución de una misma prueba los tiempos teóricos de inicio de transferencia son los mismos. Sin embargo el benchmark puede sufrir ligeros retrasos en la práctica (debido al modo en el que se inician los procesos, como se explicó en la función `EsperarCrono`, pág. 50). Por ello, antes de calcular el 95 percentil, es necesario interpolar los valores de los vectores de cada ejecución con respecto al vector de tiempos de inicio de transferencia teóricos. Esto se consigue con la instrucción `interp1`.

Obtención del uso del procesador

Para obtener los vectores con los porcentajes de uso que Apache y MySQL han hecho del procesador (MySQL sólo cuando el benchmark ha trabajado con páginas dinámicas), el programa lee línea a línea los ficheros devueltos por el monitor de procesos (en la pág. 64 se mostraba un ejemplo de este tipo de ficheros).
Mientras la línea leída no se corresponda con una marca de tiempo (*timestamp*) el programa irá sumando todos los porcentajes de consumo del procesador y almacenándolos como elementos de un vector. Cuando se leen todas las líneas se calcula el vector con el 95 percentil.

3.2.2. El programa para elaborar tablas de estadísticas

Este programa (`qestad`) calcula la media y la varianza de varios parámetros a partir de los ficheros `csv` de una prueba, y muestra los resultados por pantalla en el formato de tabla de LaTeX. Los parámetros calculados son:

1. Tiempo teórico entre peticiones. Son los valores de tiempo que el benchmark calcula mediante los parámetros que se pasan a la opción `-n`. Los valores se obtienen a partir de los datos de los ficheros `csv` etiquetados con el nombre `tiemposInicioPeticion`.

2. Tiempo real entre peticiones. Puesto que el benchmark puede sufrir ligeros retrasos en el envío de algunas peticiones, se tienen en cuenta los tiempos obtenidos en la práctica. Es un parámetro útil para comprobar que el rendimiento del programa es el adecuado (sus valores deben ser similares a los teóricos). Se obtienen a partir de los datos etiquetados como `tiempoInicioPag` en los ficheros `csv`.

3. Tamaño de las páginas. Representa los tamaños de página recibidos por el cliente, incluyendo las cabeceras HTTP. Se obtienen de los datos etiquetados como `tamanoPaginasCC`.

4. Tamaño de los objetos. Son los valores de los tamaños de los objetos que el benchmark calcula mediante los atributos pasados a la opción `-o`. Estos valores se obtienen de los datos etiquetados como `tamanoObjetos`, y sólo se calculan cuando las pruebas se han hecho con páginas estáticas.

5. Número de objetos por página. Son los valores que el benchmark calcula mediante los atributos pasados a la opción `-p`. Se obtienen a partir de los datos etiquetados como `objetosPagina`.

6. Tiempo de respuesta de páginas. Es el tiempo empleado en recibir una página completa desde que se solicita, incluyendo todos los objetos que contiene. Los valores se obtienen al restar los datos etiquetados como `tiempoTransferPag` de los etiquetados como `tiempoInicioPag` en los ficheros `csv`.

7. Tiempo de respuesta de objetos. Es una estimación del tiempo empleado en recibir un único objeto. Para su cálculo, se toma al azar una muestra lo suficientemente significativa de valores etiquetados como `tiempoTransferObj[n]` y `tiempoInicioObj[n]` y se restan. Se hace de esta forma porque si se tomaran todos los valores obtenidos de la ejecución, el tiempo de cálculo sería extremadamente grande. Estos datos sólo se calculan cuando las pruebas se han hecho con páginas estáticas.

Con estos resultados se pueden realizar algunos análisis que servirán para comprobar que los programas diseñados funcionan correctamente.

3.3. Pruebas en localhost

Los dos grupos de pruebas que se documentan en esta sección se han realizado utilizando un único ordenador que actúa como cliente y servidor al mismo tiempo. El benchmark realiza todas sus peticiones en localhost y el servidor web Apache, que se encuentra en el mismo equipo, las atiende.
Las diferencias más notables entre estas pruebas y las realizadas en red local o a través de Internet son dos: primero, que el uso del procesador del equipo se debe repartir entre los procesos del cliente y del servidor; y segundo, que estos procesos no se encuentran distribuidos en una red de ordenadores.
El ordenador empleado es un portátil EasyNote de Packard Bell, con procesador AMD Turion 64 de 1600 MHz y 512 MB de memoria RAM.
Las distribuciones y parámetros que se han utilizado en cada prueba se encuentran en el cuadro 3.1. Los valores de los parámetros se han elegido de tal forma que, para cualquiera de las cinco distribuciones, el valor medio del tiempo entre peticiones es de 0,03 segundos, el tamaño medio de los objetos es de 200 bytes, y el número medio de objetos por página es 30. En el cuadro 3.2 se resumen las medias y varianzas teóricas de los parámetros seleccionados.
El número de peticiones realizadas por cada una de las noventa ejecuciones realizadas por cada tipo de página ha sido de 20 000. Como el valor medio del tiempo entre peticiones es de 0,03 segundos para todas las pruebas, la duración de cada ejecución ha sido de, aproximadamente, diez minutos. El número de ficheros-objeto y de páginas

	Tiempo entre peticiones	Tamaño de objetos	Objetos por página
Prueba 1	Constante $c = 0{,}03$	Constante $c = 200$	Constante $c = 30$
Prueba 2	Distr. uniforme $v_{min} = 0{,}01$ $v_{max} = 0{,}05$	Distr. uniforme $v_{min} = 0$ $v_{max} = 400$	Distr. uniforme $v_{min} = 0$ $v_{max} = 60$
Prueba 3	Distr. exponencial $\mu = 0{,}03$	Distr. exponencial $\mu = 200$	Distr. exponencial $\mu = 30$
Prueba 4	Distr. exponencial $\mu = 0{,}03$	Distr. Pareto $x_m = 57$, $k = 1{,}4$	Constante $c = 1$
Prueba 5	Distr. Pareto $x_m = 0{,}018$, $k = 2{,}5$	Distr. Pareto $x_m = 57$, $k = 1{,}4$	Constante $c = 1$
Prueba 6	Distr. hiperexponencial $\mu_1 = 0{,}025$, $\mu_2 = 0{,}1$, $p_1 = 0{,}9333$	Distr. Pareto $x_m = 57$, $k = 1{,}4$	Constante $c = 1$
Prueba 7	Distr. exponencial $\mu = 0{,}03$	Distr. lognormal $\mu = 200$, $\sigma^2 = 40000$	Distr. Pareto $x_m = 18$, $k = 2{,}5$
Prueba 8	Distr. Pareto $x_m = 0{,}018$, $k = 2{,}5$	Distr. lognormal $\mu = 200$, $\sigma^2 = 40000$	Distr. Pareto $x_m = 18$, $k = 2{,}5$
Prueba 9	Distr. hiperexponencial $\mu_1 = 0{,}025$, $\mu_2 = 0{,}1$, $p_1 = 0{,}9333$	Distr. lognormal $\mu = 200$, $\sigma^2 = 40000$	Distr. Pareto $x_m = 18$, $k = 2{,}5$

Cuadro 3.1: Distribuciones y atributos usados en las pruebas en localhost.

	Tiempo peticiones	Tamaño objetos	Obj./Pág.
Constante	$\mu = 0{,}03$ $\sigma^2 = 0$	$\mu = 200$ $\sigma^2 = 0$	$\mu = 30$ $\sigma^2 = 0$
D. Uniforme	$\mu = 0{,}03$ $\sigma^2 = 1{,}333 \cdot 10^{-4}$	$\mu = 200$ $\sigma^2 = 1{,}333 \cdot 10^{4}$	$\mu = 30$ $\sigma^2 = 300$
D. Exponencial	$\mu = 0{,}03$ $\sigma^2 = 9 \cdot 10^{-4}$	$\mu = 200$ $\sigma^2 = 4 \cdot 10^{4}$	$\mu = 30$ $\sigma^2 = 900$
D. Pareto	$\mu = 0{,}03$ $\sigma^2 = 7{,}2 \cdot 10^{-4}$	$\mu \approx 200$ $\sigma^2 = -$	$\mu = 30$ $\sigma^2 = 720$
D. Hiperexpon.	$\mu \approx 0{,}03$ $\sigma^2 \approx 1{,}6 \cdot 10^{-3}$	—	—
D. Lognormal	—	$\mu = 200$ $\sigma^2 = 4 \cdot 10^{4}$	—

Cuadro 3.2: Valores medios y varianzas teóricas de las pruebas en localhost.

web generadas ha sido de 50 en todas las pruebas. El valor de la semilla aleatoria utilizada es, como ya se indicó anteriormente, $s = 241181$.

3.3.1. Páginas estáticas

Las gráficas con los resultados de cada prueba (el 95 percentil del tiempo de respuesta y del porcentaje de procesador usado por Apache, ambas en función del tiempo de ejecución) se muestran en las figuras 3.1 a 3.9. Las estadísticas devueltas por el programa `qestad` se encuentran en los cuadros 3.3 a 3.11.

De las gráficas generadas se comprueba en todos los casos que, para el 95 % de las ejecuciones, el tiempo de respuesta máximo se sitúa en torno a los 0,3 segundos. Las gráficas de las pruebas 7, 8 y 9, en las que el número de objetos por página se define mediante la distribución de Pareto y el tamaño de los objetos mediante la distribución lognormal, muestran varios picos de entre 0,4 y 0,6 segundos.

Por otro lado, las pruebas 4, 5 y 6, en las que todas las páginas web cuentan con un único objeto, son las que ofrecen menor tiempo medio de respuesta y uso de procesador. Como se ve en las figuras 3.4, 3.5 y 3.6, los picos máximos en el tiempo de respuesta raramente sobrepasan los 200 milisegundos.

En general, en todas las pruebas se observa que el tiempo de respuesta tiende a aumentar a medida que transcurre el tiempo de ejecución. En algunos casos, este incremento viene acompañado de un uso del procesador más acusado, sobre todo hacia el final de la prueba. Este comportamiento se puede ver claramente, por ejemplo, en las figuras correspondientes a las pruebas 1 y 7.

En ningún caso se observa que el proceso de Apache supere el 20 % de consumo de la CPU (en las pruebas con otras topologías de red es frecuente que esto sí ocurra).

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,03	$2{,}3083 \cdot 10^{-28}$
Tiempo real entre peticiones (s)	0,0300007	$3{,}23213 \cdot 10^{-6}$
Tamaño de las páginas (bytes)	1704,9	0,000663486
Tamaño de los objetos (bytes)	200	0
Número de objetos por página	30	0
Tiempo de respuesta de páginas (s)	0,0178449	0,000232823
Tiempo de respuesta de objetos (s)	0,000605952	$1{,}99875 \cdot 10^{-6}$

Cuadro 3.3: Estadísticas de la prueba 1 para páginas estáticas en localhost.

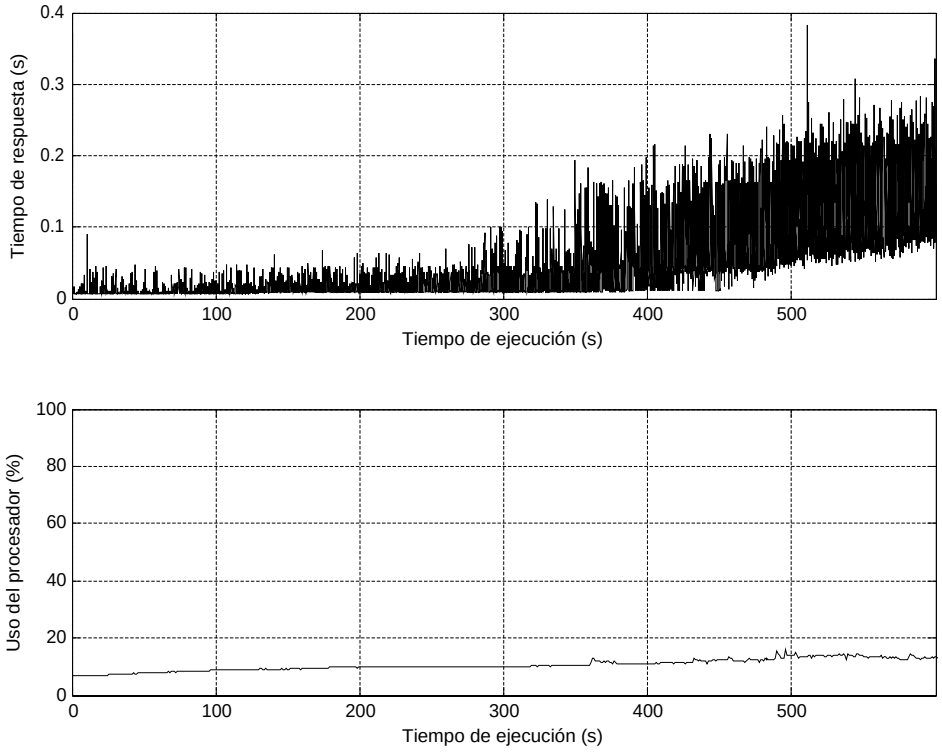

Figura 3.1: Gráficas de la prueba 1 para páginas estáticas en localhost.

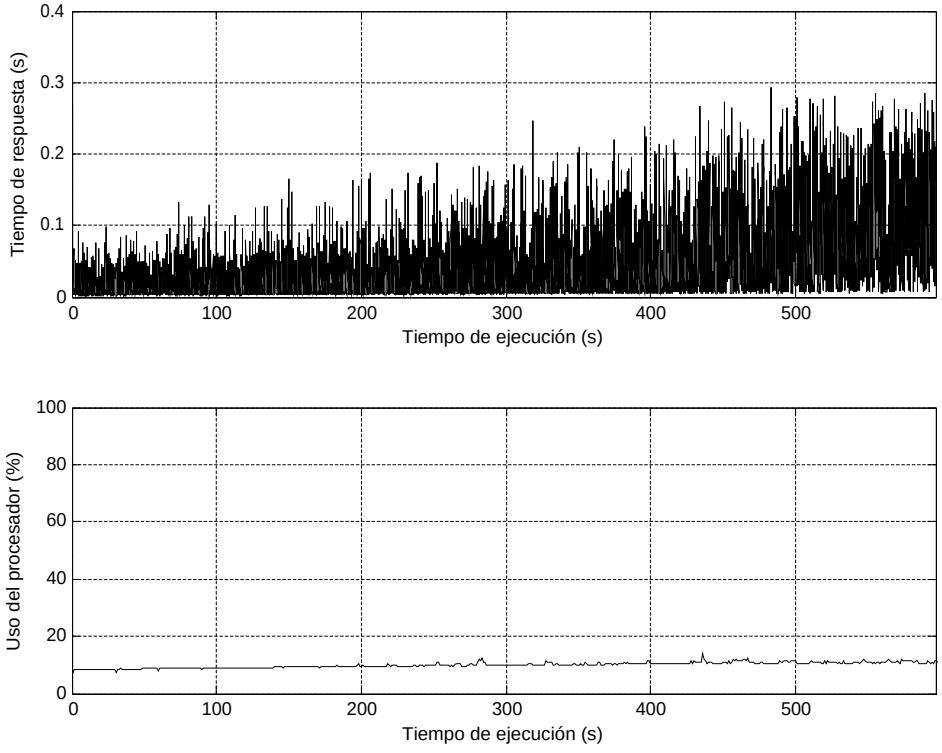

Figura 3.2: Gráficas de la prueba 2 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298286	0,000132545
Tiempo real entre peticiones (s)	0,0298286	0,000111669
Tamaño de las páginas (bytes)	1553,72	433748
Tamaño de los objetos (bytes)	188,64	11577,9
Número de objetos por página	25,98	345
Tiempo de respuesta de páginas (s)	0,0192246	0,000256393
Tiempo de respuesta de objetos (s)	0,000495191	$7{,}1518 \cdot 10^{-7}$

Cuadro 3.4: Estadísticas de la prueba 2 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0303021	0,000700411
Tamaño de las páginas (bytes)	1556,35	862404
Tamaño de los objetos (bytes)	192,96	22527,5
Número de objetos por página	26,58	703,596
Tiempo de respuesta de páginas (s)	0,0266395	0,000458168
Tiempo de respuesta de objetos (s)	0,000692663	$6{,}30769 \cdot 10^{-7}$

Cuadro 3.5: Estadísticas de la prueba 3 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302902	0,000819834
Tamaño de las páginas (bytes)	511,901	0,000647542
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,00919871	$5{,}18069 \cdot 10^{-5}$
Tiempo de respuesta de objetos (s)	0,0012278	$2{,}26653 \cdot 10^{-6}$

Cuadro 3.6: Estadísticas de la prueba 4 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299416	0,00042371
Tamaño de las páginas (bytes)	512,001	0,000713279
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,00370546	$1{,}17321 \cdot 10^{-5}$
Tiempo de respuesta de objetos (s)	0,000476317	$2{,}40347 \cdot 10^{-7}$

Cuadro 3.7: Estadísticas de la prueba 5 para páginas estáticas en localhost.

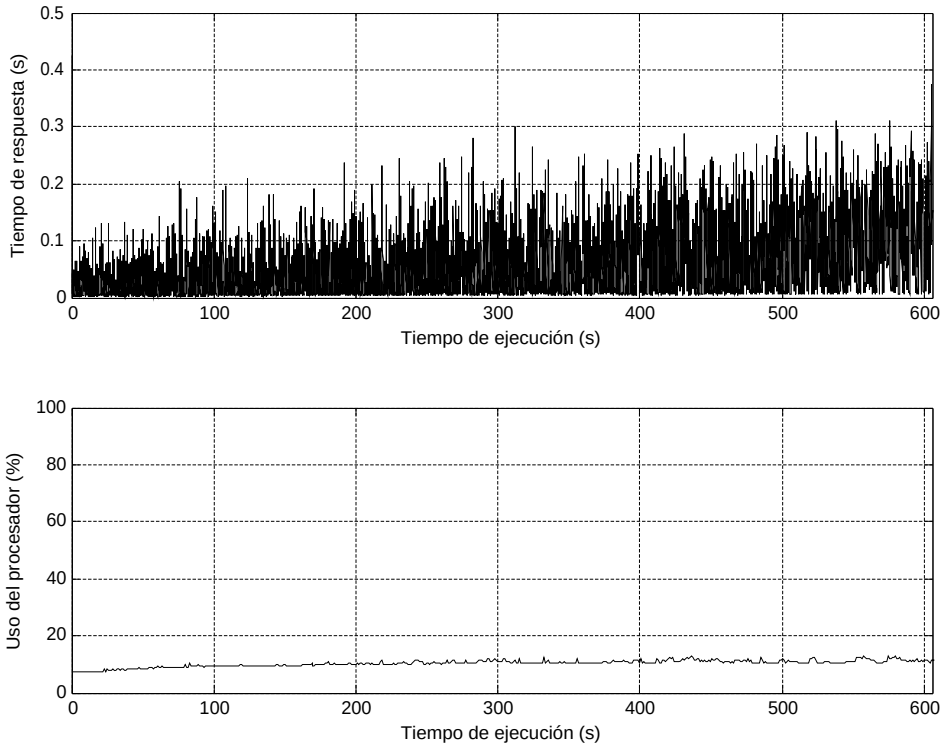

Figura 3.3: Gráficas de la prueba 3 para páginas estáticas en localhost.

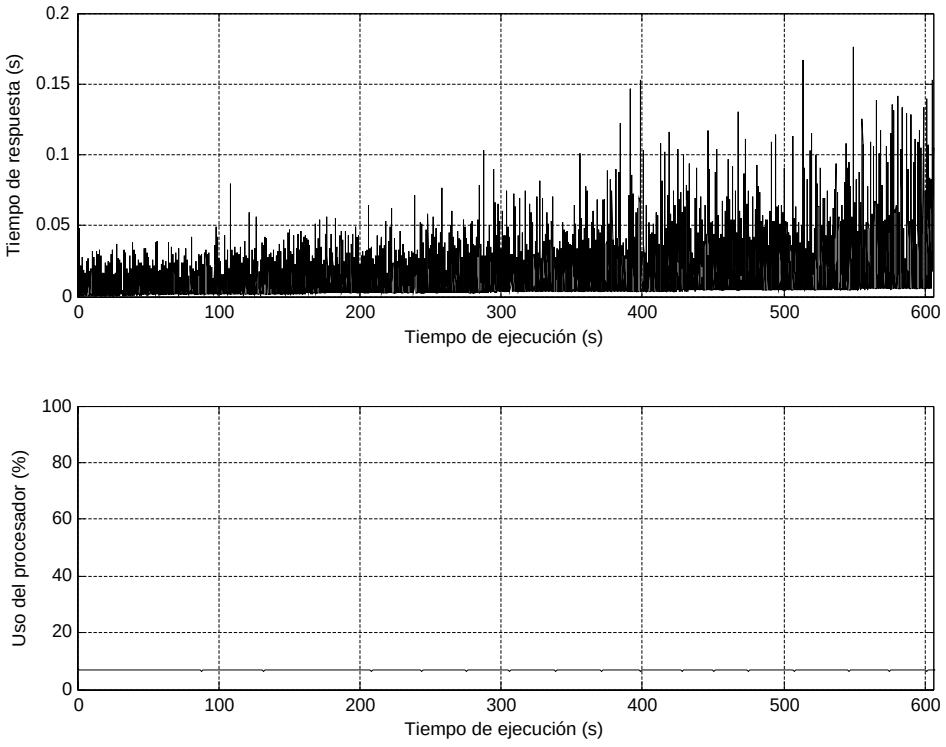

Figura 3.4: Gráficas de la prueba 4 para páginas estáticas en localhost.

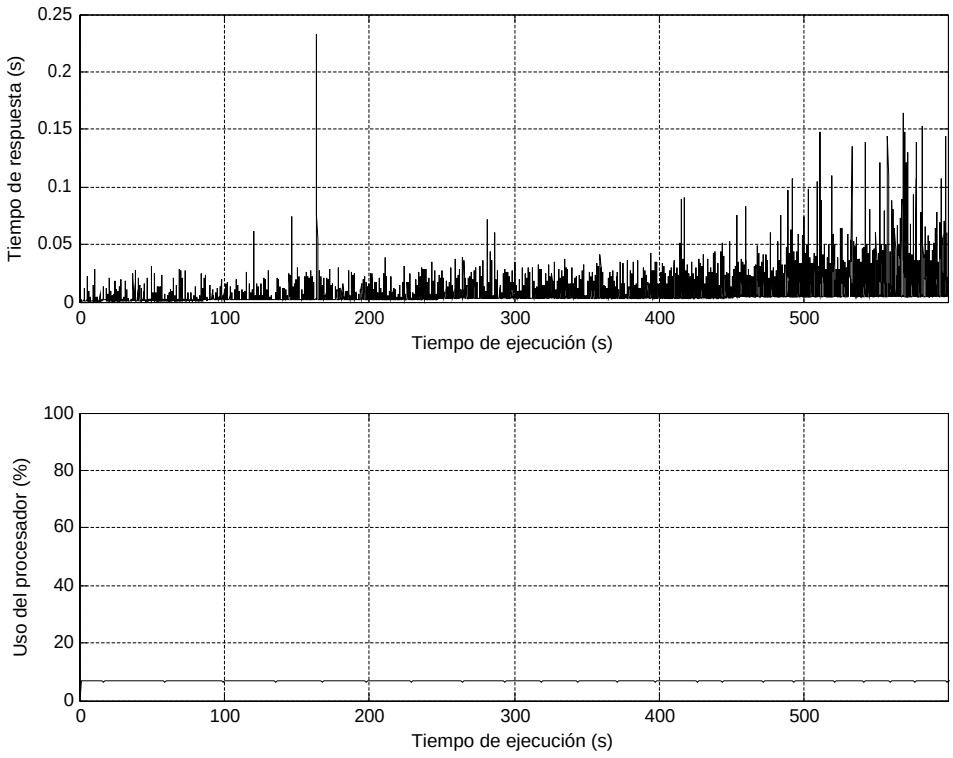

Figura 3.5: Gráficas de la prueba 5 para páginas estáticas en localhost.

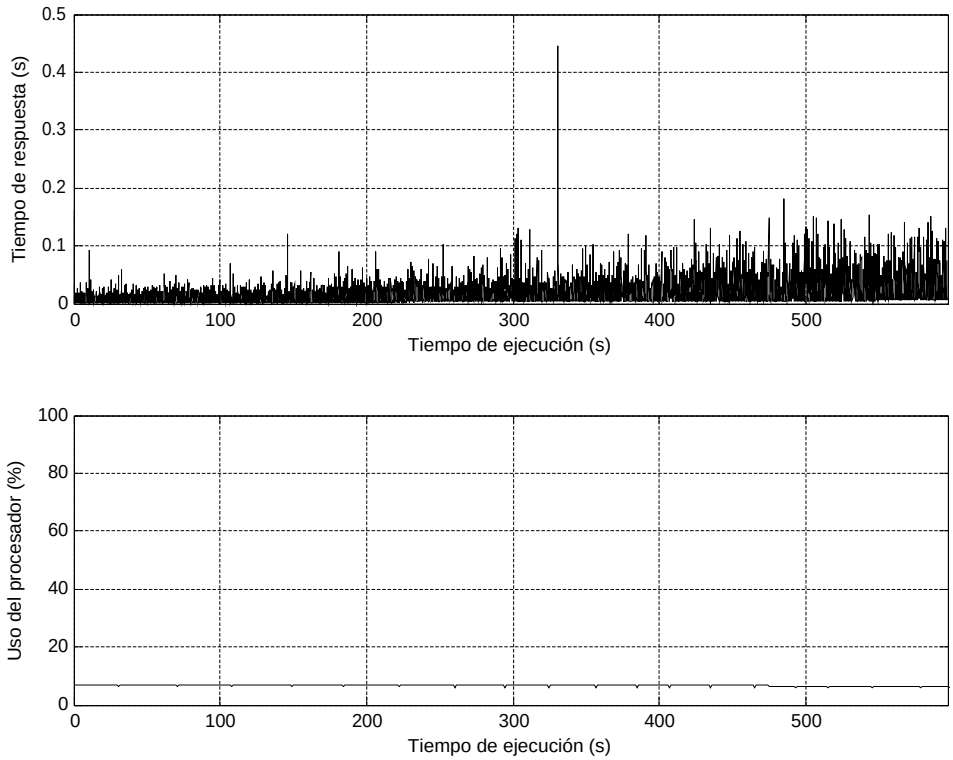

Figura 3.6: Gráficas de la prueba 6 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00148852
Tamaño de las páginas (bytes)	512,001	0,000659393
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0101754	$6{,}19962 \cdot 10^{-5}$
Tiempo de respuesta de objetos (s)	0,00183357	$7{,}65138 \cdot 10^{-6}$

Cuadro 3.8: Estadísticas de la prueba 6 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0303014	0,000674293
Tamaño de las páginas (bytes)	1682,78	412233
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	0,0305051	0,000436217
Tiempo de respuesta de objetos (s)	0,000614086	$9{,}93832 \cdot 10^{-7}$

Cuadro 3.9: Estadísticas de la prueba 7 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299447	0,000401807
Tamaño de las páginas (bytes)	1682,78	412233
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	0,0258999	0,00039019
Tiempo de respuesta de objetos (s)	0,000649259	$2{,}64699 \cdot 10^{-6}$

Cuadro 3.10: Estadísticas de la prueba 8 para páginas estáticas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00126057
Tamaño de las páginas (bytes)	1685,45	408126
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	0,033852	0,000503861
Tiempo de respuesta de objetos (s)	0,000609967	$6{,}37217 \cdot 10^{-7}$

Cuadro 3.11: Estadísticas de la prueba 9 para páginas estáticas en localhost.

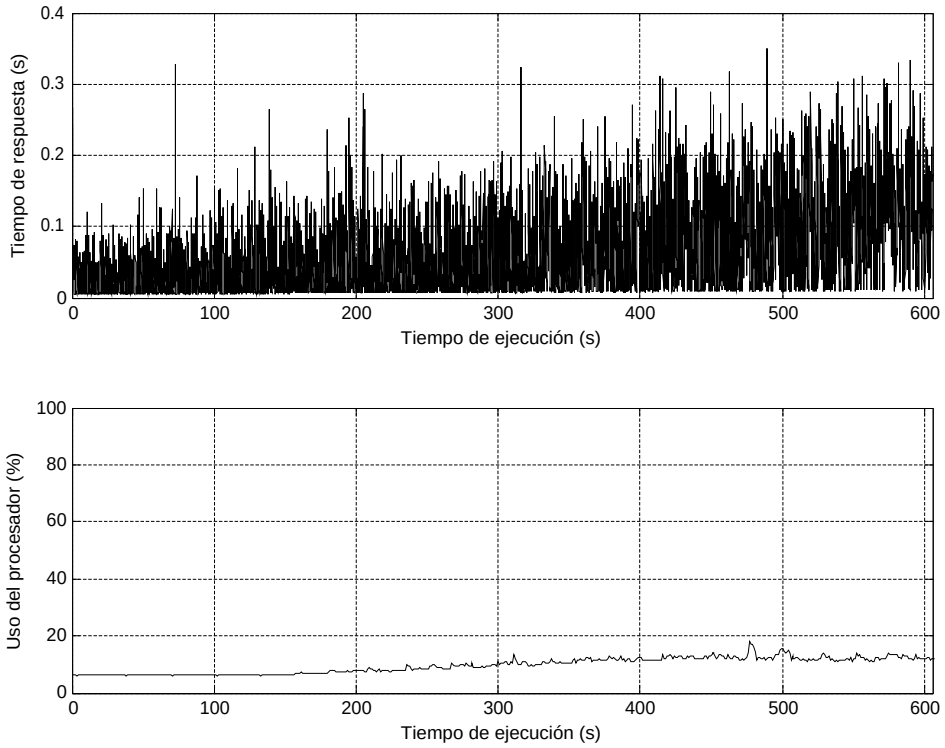

Figura 3.7: Gráficas de la prueba 7 para páginas estáticas en localhost.

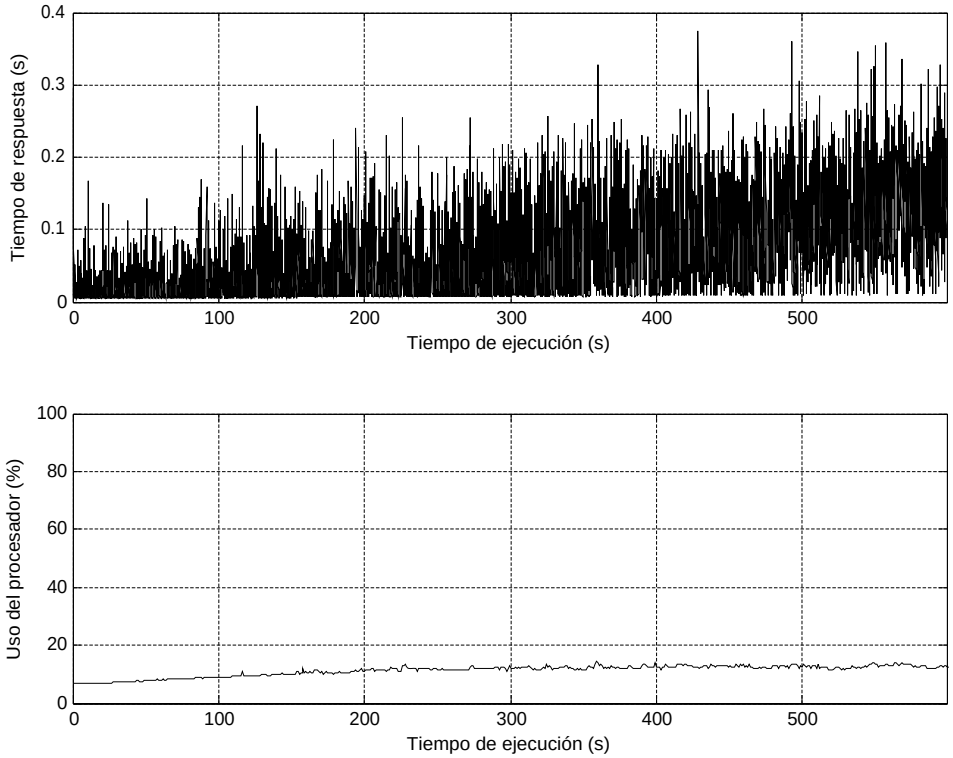

Figura 3.8: Gráficas de la prueba 8 para páginas estáticas en localhost.

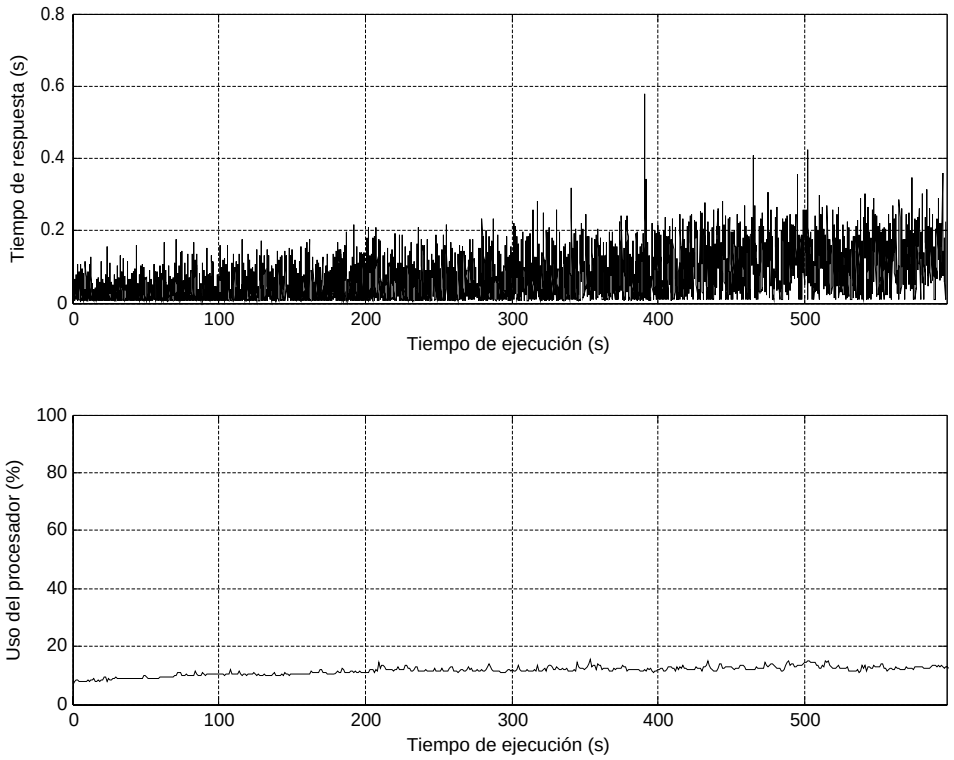

Figura 3.9: Gráficas de la prueba 9 para páginas estáticas en localhost.

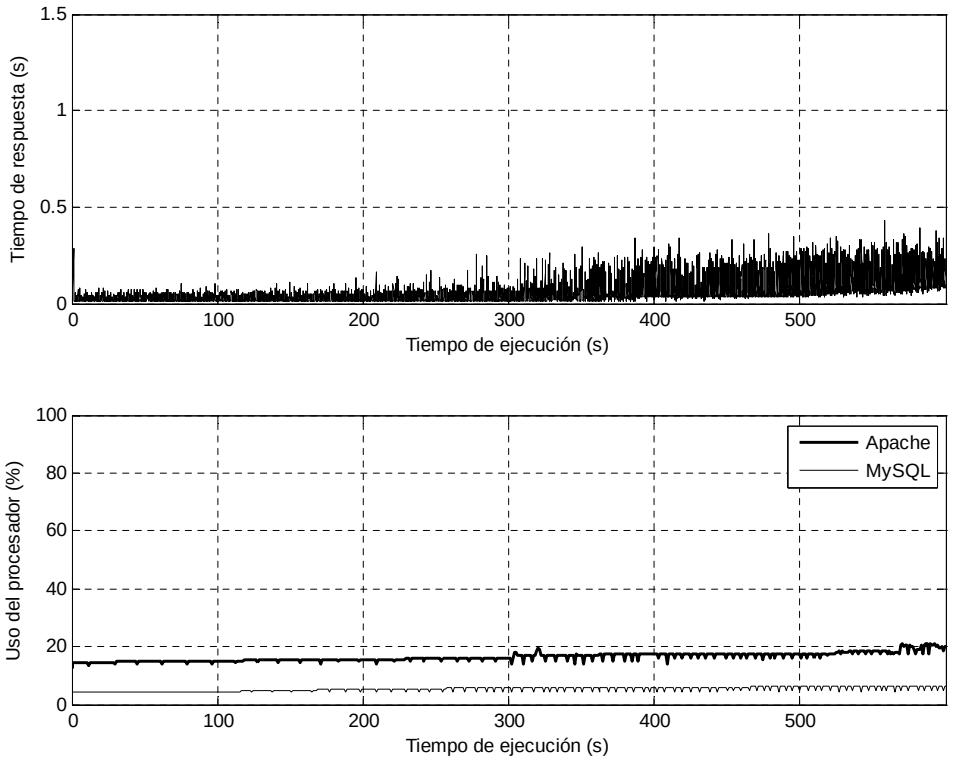

Figura 3.10: Gráficas de la prueba 1 para páginas dinámicas en localhost.

3.3.2. Páginas dinámicas

Las gráficas con los resultados de las pruebas se pueden encontrar en las figuras 3.10 a 3.18, y las tablas con los datos estadísticos devueltos por Matlab, se muestran en los cuadros 3.12 a 3.20.

Los resultados son bastante similares a los obtenidos con páginas estáticas. El tiempo de respuesta máximo (para el 95 % de las ejecuciones) se sitúa en torno a los $0,4$ segundos. Como antes, las pruebas 4, 5 y 6, en las que todas sus páginas cuentan con un único objeto, son las que ofrecen menor tiempo medio de respuesta (en pocas ocasiones se sobrepasan los 150 milisegundos) y uso de procesador. En todas las pruebas se observa que el tiempo de respuesta tiende a aumentar a medida que transcurre el tiempo de ejecución.

El procesador raramente supera el 20 % de uso para Apache y no llega al 10 % con MySQL, por lo que, en conjunto, nunca se sobrepasa el 30 % de uso.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,03	$2{,}3083 \cdot 10^{-28}$
Tiempo real entre peticiones (s)	0,0300013	$3{,}51182 \cdot 10^{-6}$
Tamaño de las páginas (bytes)	6626	0
Número de objetos por página	30	0
Tiempo de respuesta de páginas (s)	0,0211935	0,000280392

Cuadro 3.12: Estadísticas de la prueba 1 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298286	0,000132545
Tiempo real entre peticiones (s)	0,0298286	0,000115522
Tamaño de las páginas (bytes)	6096,51	$1{,}17263 \cdot 10^{7}$
Número de objetos por página	27,78	260,134
Tiempo de respuesta de páginas (s)	0,0276577	0,000436756

Cuadro 3.13: Estadísticas de la prueba 2 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0303077	0,000721805
Tamaño de las páginas (bytes)	6376,74	$2{,}64078 \cdot 10^{7}$
Número de objetos por página	28,58	508,738
Tiempo de respuesta de páginas (s)	0,0408687	0,00087923

Cuadro 3.14: Estadísticas de la prueba 3 para páginas dinámicas en localhost.

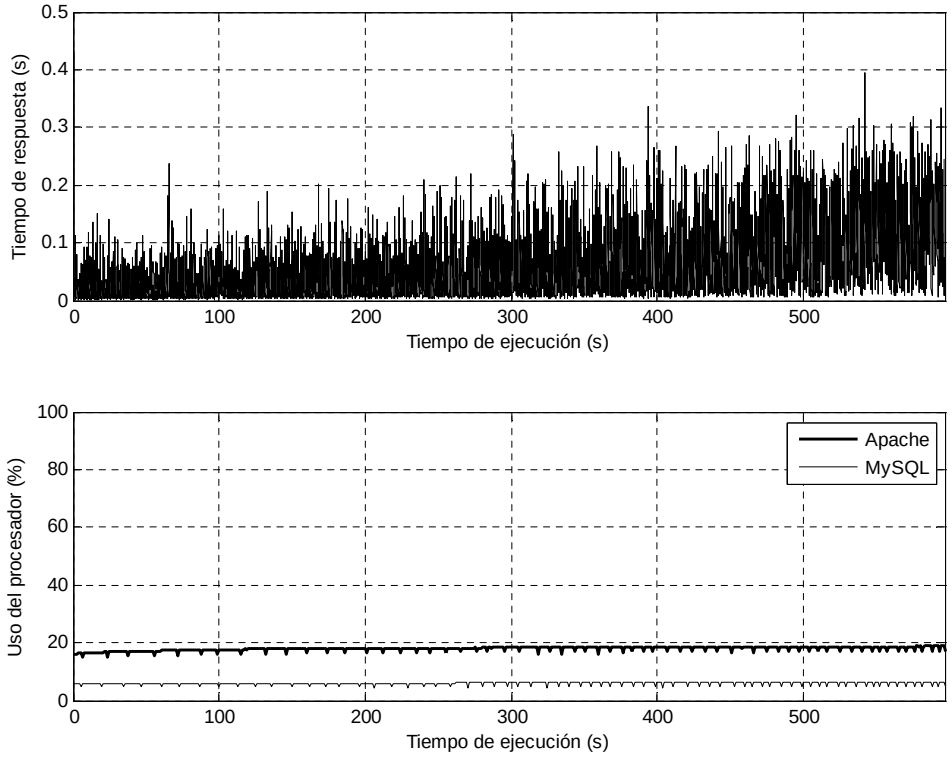

Figura 3.11: Gráficas de la prueba 2 para páginas dinámicas en localhost.

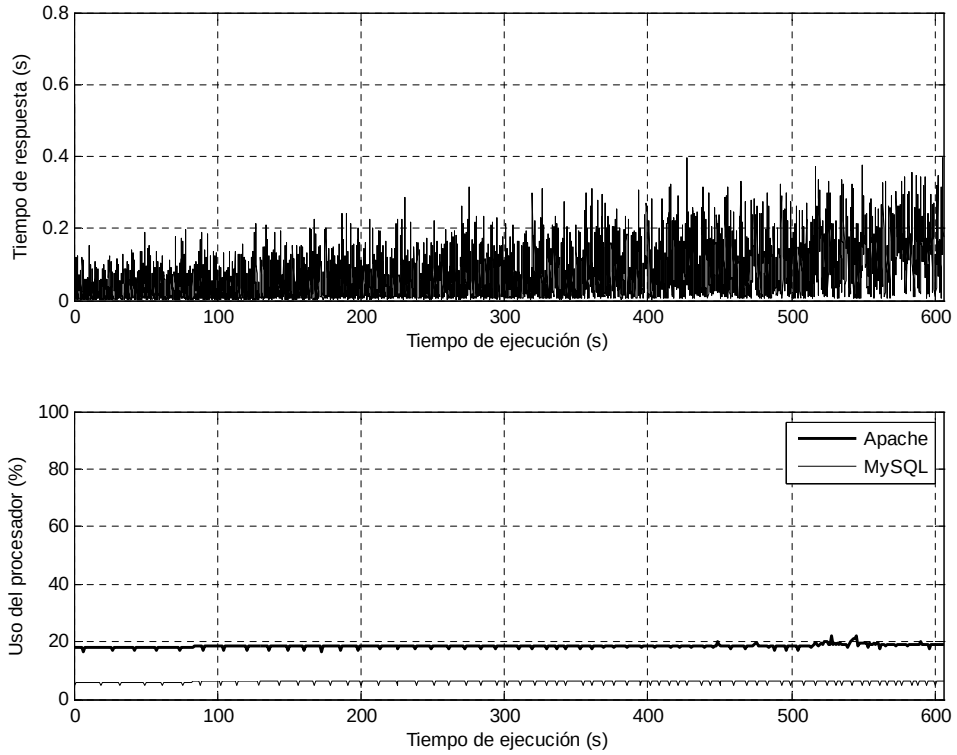

Figura 3.12: Gráficas de la prueba 3 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302901	0,000830033
Tamaño de las páginas (bytes)	602,691	185853
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0138686	0,000117347

Cuadro 3.15: Estadísticas de la prueba 4 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299417	0,000423674
Tamaño de las páginas (bytes)	602,691	185853
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0054825	$2{,}81358 \cdot 10^{-5}$

Cuadro 3.16: Estadísticas de la prueba 5 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298299	0,00150329
Tamaño de las páginas (bytes)	597,91	177798
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0148755	0,000130256

Cuadro 3.17: Estadísticas de la prueba 6 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0303021	0,000744054
Tamaño de las páginas (bytes)	6098,45	$5{,}2214 \cdot 10^{6}$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,0378965	0,000608919

Cuadro 3.18: Estadísticas de la prueba 7 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299467	0,000409353
Tamaño de las páginas (bytes)	6098,45	$5{,}2214 \cdot 10^{6}$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,0273983	0,000429868

Cuadro 3.19: Estadísticas de la prueba 8 para páginas dinámicas en localhost.

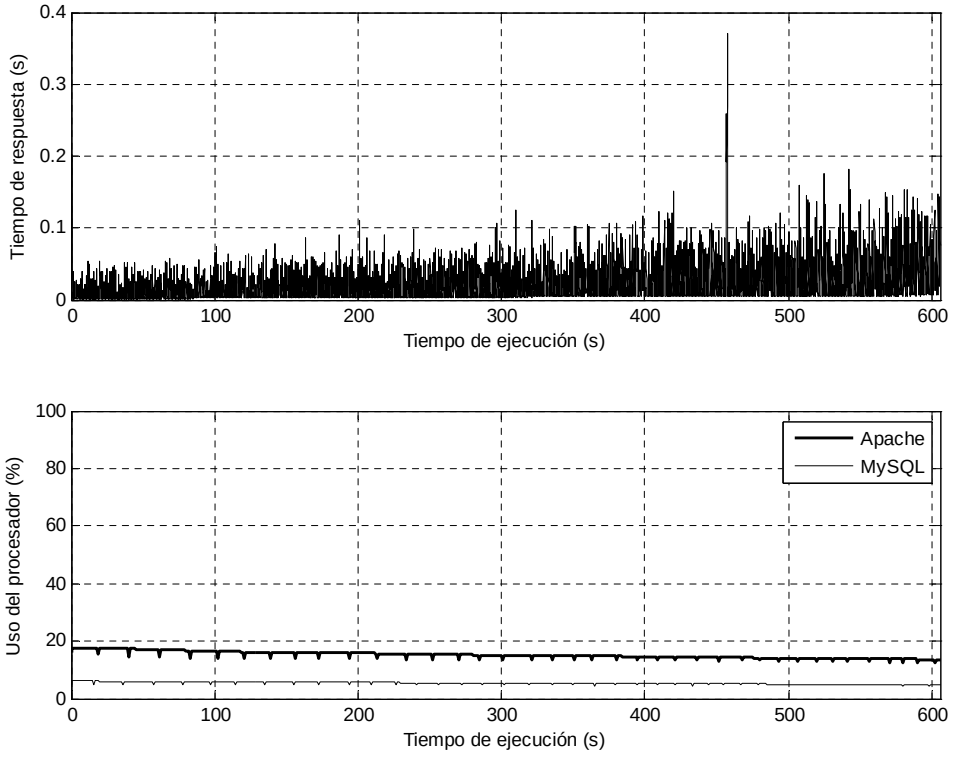

Figura 3.13: Gráficas de la prueba 4 para páginas dinámicas en localhost.

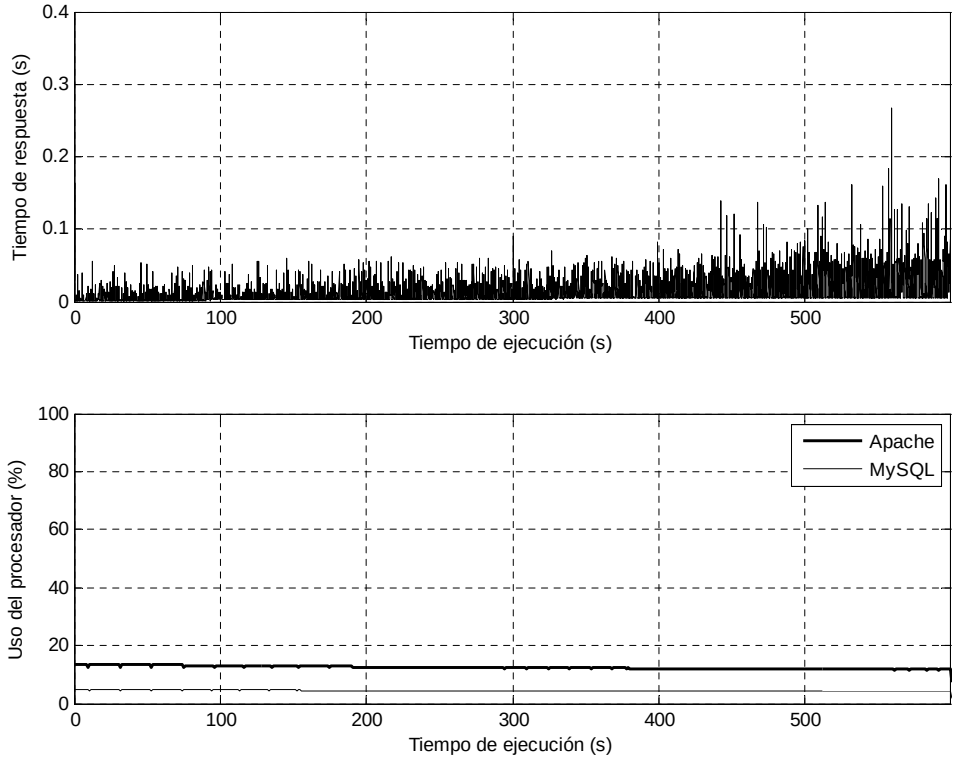

Figura 3.14: Gráficas de la prueba 5 para páginas dinámicas en localhost.

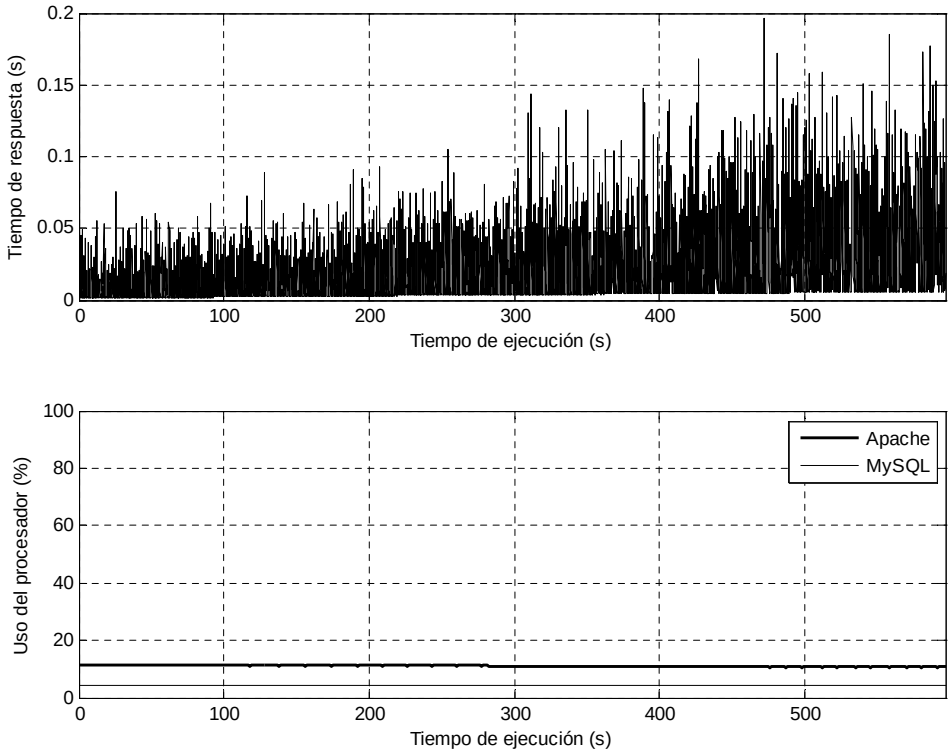

Figura 3.15: Gráficas de la prueba 6 para páginas dinámicas en localhost.

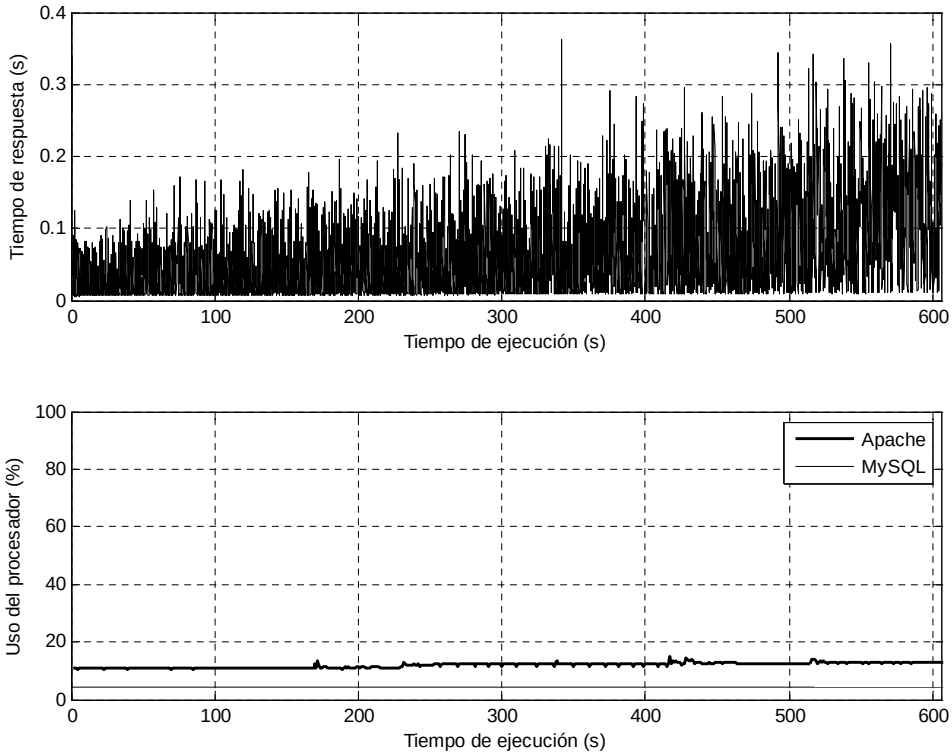

Figura 3.16: Gráficas de la prueba 7 para páginas dinámicas en localhost.

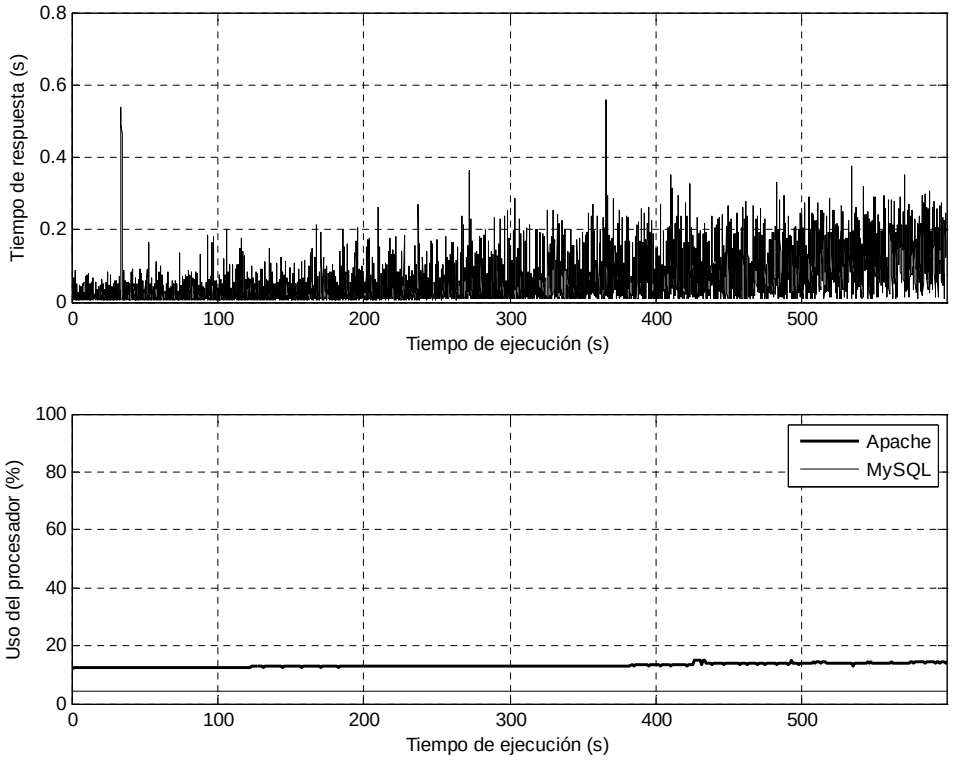

Figura 3.17: Gráficas de la prueba 8 para páginas dinámicas en localhost.

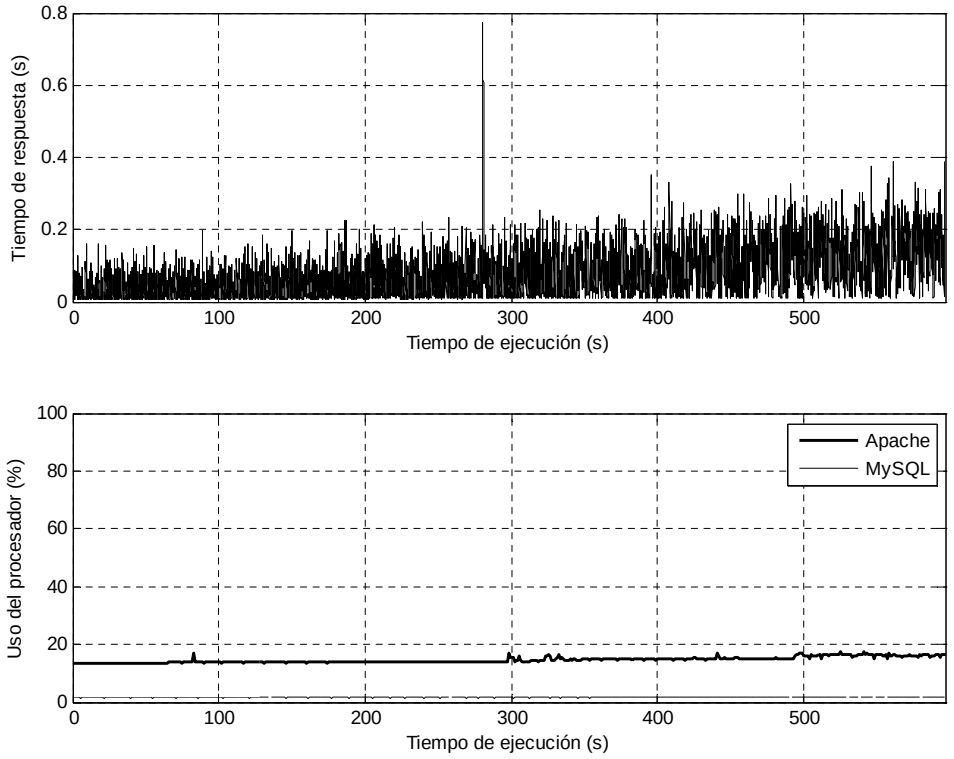

Figura 3.18: Gráficas de la prueba 9 para páginas dinámicas en localhost.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00131733
Tamaño de las páginas (bytes)	6115,84	$5{,}24249 \cdot 10^6$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,0437433	0,000749099

Cuadro 3.20: Estadísticas de la prueba 9 para páginas dinámicas en localhost.

3.4. Pruebas en red local

Las pruebas que se comentan en las siguientes secciones se han realizado utilizando dos ordenadores conectados en una red de área local (LAN), tal como se muestra en el esquema de la siguiente figura:

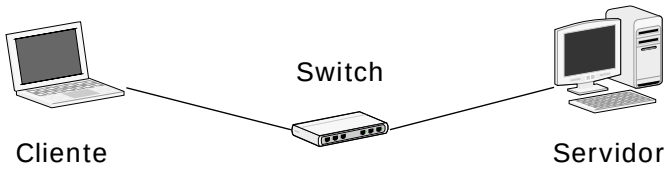

Figura 3.19: Topología usada con las pruebas en red local.

El ordenador que hace de cliente es el portátil descrito en la sección anterior, mientras que el equipo servidor es un ordenador de sobremesa Compaq Presario, con un procesador Intel Pentium D a 3000 MHz y 2 GB de memoria RAM. Los ordenadores se conectan entre sí mediante un *switch* Fast Ethernet.

Las distribuciones y atributos que se han utilizado en cada prueba son los mismos que los utilizados en localhost (véase el cuadro 3.1). Por ello, los valores medios y las varianzas de dichos parámetros también son los que se muestran en el cuadro 3.2.

El número de peticiones que se han realizado en cada ejecución ha sido de 20 000. El valor medio del tiempo entre peticiones es de 0,03 segundos, por lo tanto, la duración de cada ejecución ha sido de aproximadamente diez minutos. Se han generado 50 ficheros-objeto y 50 páginas web en todas las pruebas. El valor de la semilla aleatoria ha sido $s = 241181$.

3.4.1. Páginas estáticas

Las gráficas con los resultados de las pruebas se muestran en las figuras 3.20 a 3.28, y las tablas con los datos estadísticos se encuentran en los cuadros 3.21 a 3.29.

Si se comparan las tablas de resultados de estas pruebas con las realizadas en localhost, se aprecia un incremento significativo en los tiempos de respuesta de páginas y objetos (en algunos casos se multiplican por dos o tres).

En las gráficas con el 95 percentil de los tiempos de respuesta, sin embargo, se mantiene el valor máximo del tiempo de respuesta, que se sitúa en torno a los 0,4 segundos. Como en los casos anteriores, esto es más acusado para las pruebas 7, 8 y 9, donde se combinan las distribuciones lognormal y de Pareto para definir las páginas web; y menos acusado para las pruebas 4, 5 y 6, donde las páginas sólo cuentan con un objeto.

El proceso de Apache hace un uso más acusado del procesador en los minutos cercanos a la finalización de las ejecuciones, en los que suele superar el 20 %. Esta tendencia se invierte en las pruebas 4, 5 y 6, donde no alcanza el 10 %.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,03	$2{,}3083 \cdot 10^{-28}$
Tiempo real entre peticiones (s)	0,03	$7{,}54601 \cdot 10^{-7}$
Tamaño de las páginas (bytes)	1703,9	0,000115948
Tamaño de los objetos (bytes)	200	0
Número de objetos por página	30	0
Tiempo de respuesta de páginas (s)	0,0385415	0,00063738
Tiempo de respuesta de objetos (s)	0,00129558	$2{,}92563 \cdot 10^{-6}$

Cuadro 3.21: Estadísticas de la prueba 1 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298286	0,000132545
Tiempo real entre peticiones (s)	0,0298286	0,000133789
Tamaño de las páginas (bytes)	1552,72	433748
Tamaño de los objetos (bytes)	188,64	11577,9
Número de objetos por página	25,98	345
Tiempo de respuesta de páginas (s)	0,0424144	0,00149361
Tiempo de respuesta de objetos (s)	0,00128813	$1{,}96079 \cdot 10^{-6}$

Cuadro 3.22: Estadísticas de la prueba 2 para páginas estáticas en red local.

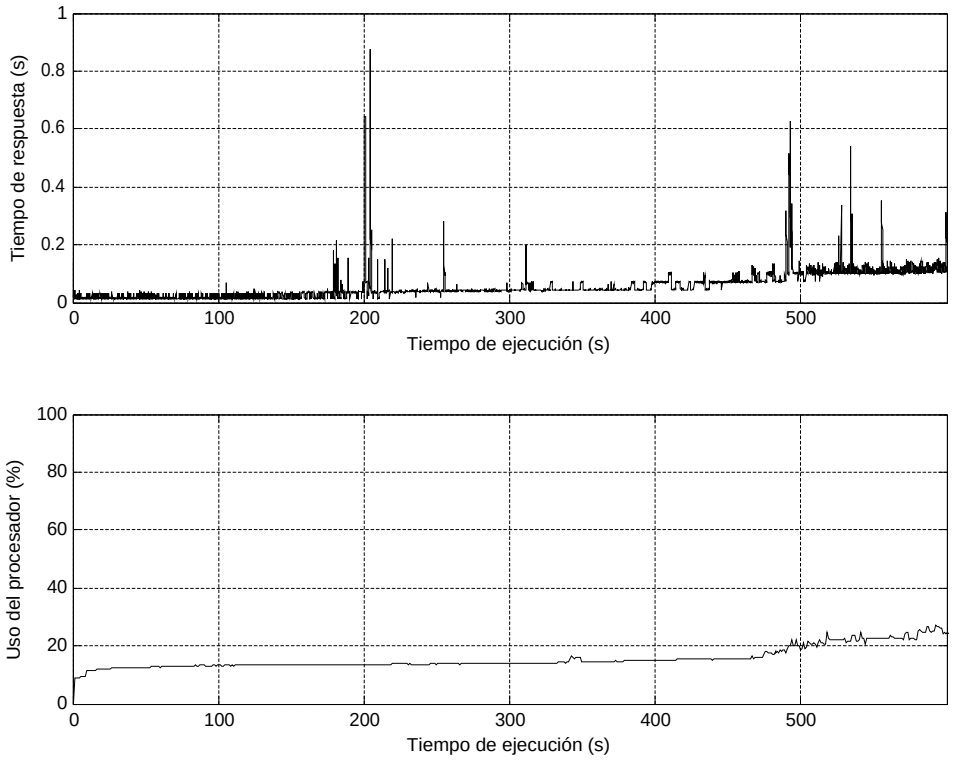

Figura 3.20: Gráficas de la prueba 1 para páginas estáticas en red local.

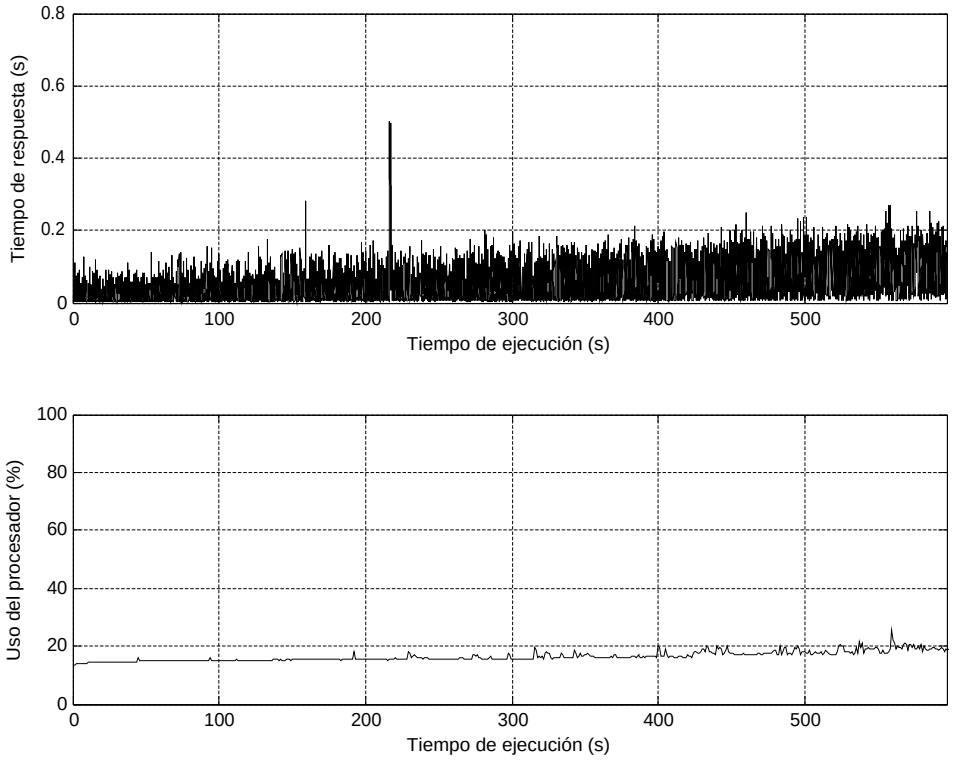

Figura 3.21: Gráficas de la prueba 2 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302906	0,000872242
Tamaño de las páginas (bytes)	1555,35	862404
Tamaño de los objetos (bytes)	192,96	22527,5
Número de objetos por página	26,58	703,596
Tiempo de respuesta de páginas (s)	0,0571862	0,00286164
Tiempo de respuesta de objetos (s)	0,00204794	$4{,}35424 \cdot 10^{-6}$

Cuadro 3.23: Estadísticas de la prueba 3 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302894	0,000882387
Tamaño de las páginas (bytes)	511	0,00011397
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0155192	0,000260231
Tiempo de respuesta de objetos (s)	0,0033002	$1{,}47962 \cdot 10^{-5}$

Cuadro 3.24: Estadísticas de la prueba 4 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299417	0,000425234
Tamaño de las páginas (bytes)	510,9	0,000259819
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,003441	$1{,}35318 \cdot 10^{-5}$
Tiempo de respuesta de objetos (s)	0,000694017	$2{,}29321 \cdot 10^{-7}$

Cuadro 3.25: Estadísticas de la prueba 5 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00156925
Tamaño de las páginas (bytes)	510,9	$7{,}9988 \cdot 10^{-6}$
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0181117	0,000336226
Tiempo de respuesta de objetos (s)	0,00438502	$7{,}43561 \cdot 10^{-5}$

Cuadro 3.26: Estadísticas de la prueba 6 para páginas estáticas en red local.

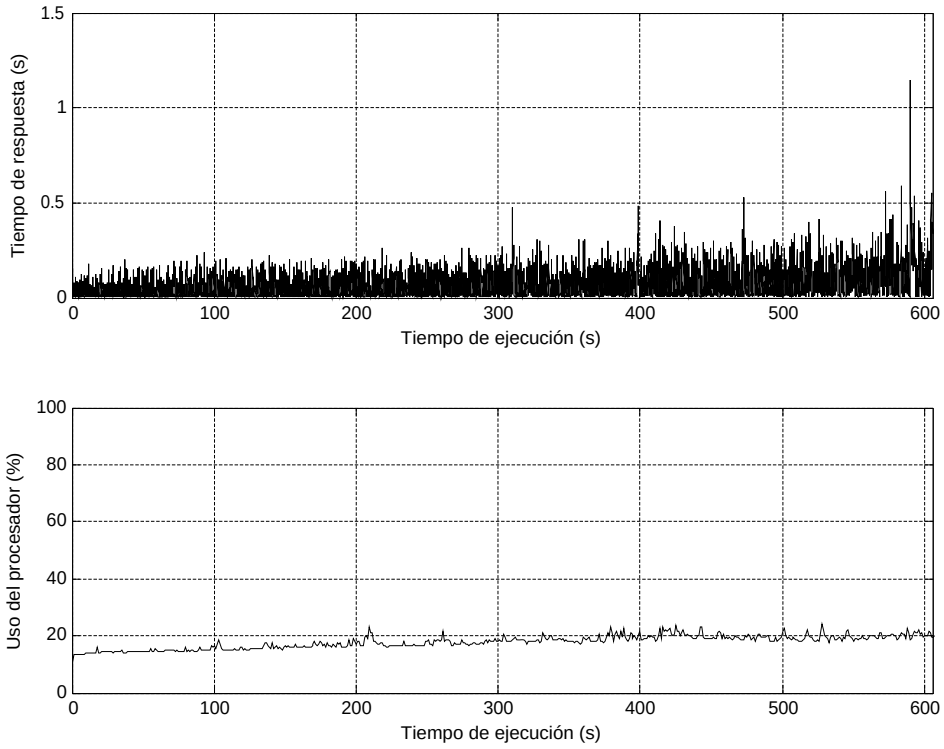

Figura 3.22: Gráficas de la prueba 3 para páginas estáticas en red local.

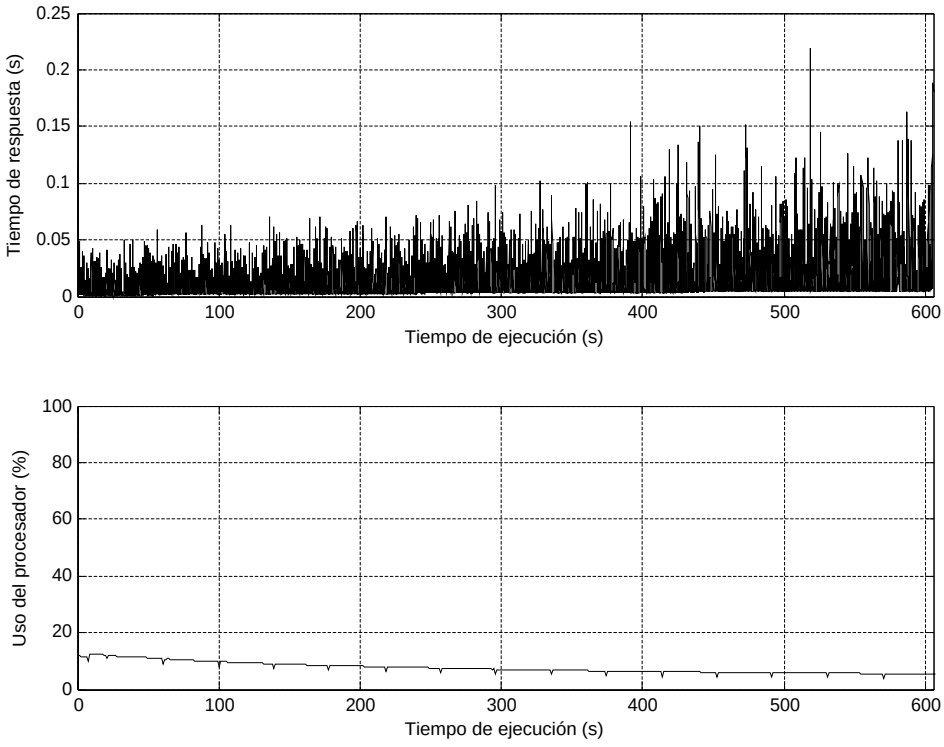

Figura 3.23: Gráficas de la prueba 4 para páginas estáticas en red local.

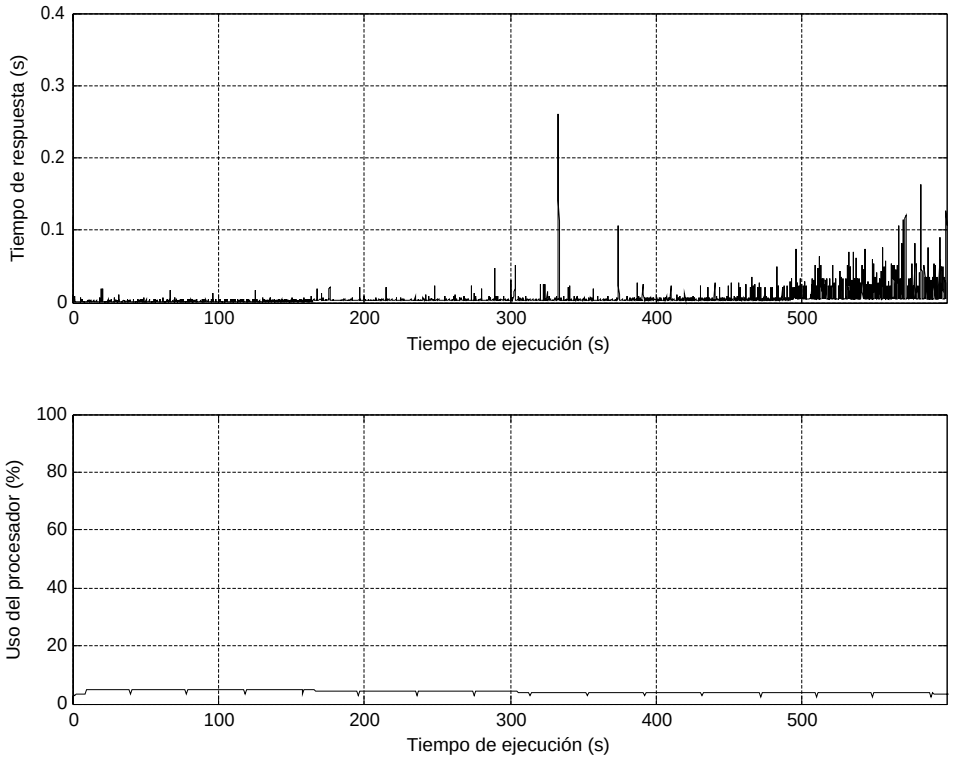

Figura 3.24: Gráficas de la prueba 5 para páginas estáticas en red local.

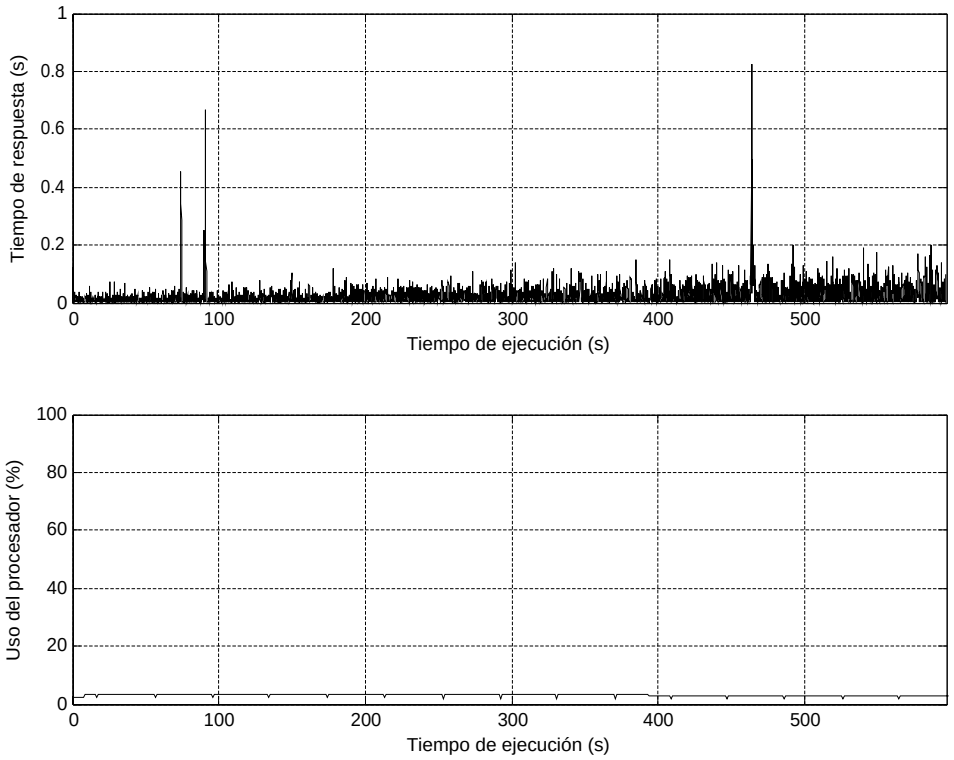

Figura 3.25: Gráficas de la prueba 6 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302901	0,000872252
Tamaño de las páginas (bytes)	1681,88	412234
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	0,0662752	0,0024742
Tiempo de respuesta de objetos (s)	0,00178894	$6{,}21938 \cdot 10^{-6}$

Cuadro 3.27: Estadísticas de la prueba 7 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299419	0,000425306
Tamaño de las páginas (bytes)	1681,88	412233
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	0,0609282	0,00196238
Tiempo de respuesta de objetos (s)	0,00187694	$4{,}11278 \cdot 10^{-6}$

Cuadro 3.28: Estadísticas de la prueba 8 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00154976
Tamaño de las páginas (bytes)	1684,35	408126
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	0,0774674	0,00308547
Tiempo de respuesta de objetos (s)	0,00173373	$4{,}16863 \cdot 10^{-6}$

Cuadro 3.29: Estadísticas de la prueba 9 para páginas estáticas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,03	$2{,}3083 \cdot 10^{-28}$
Tiempo real entre peticiones (s)	0,03	$2{,}48602 \cdot 10^{-7}$
Tamaño de las páginas (bytes)	6626	0
Número de objetos por página	30	0
Tiempo de respuesta de páginas (s)	0,00985564	$1{,}69431 \cdot 10^{-6}$

Cuadro 3.30: Estadísticas de la prueba 1 para páginas dinámicas en red local.

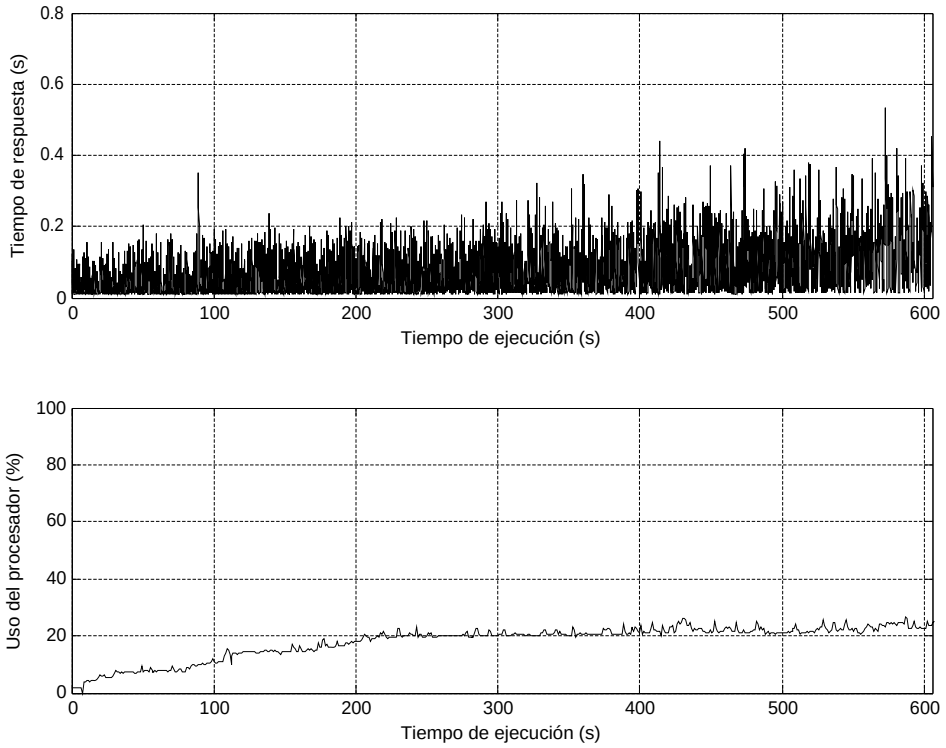

Figura 3.26: Gráficas de la prueba 7 para páginas estáticas en red local.

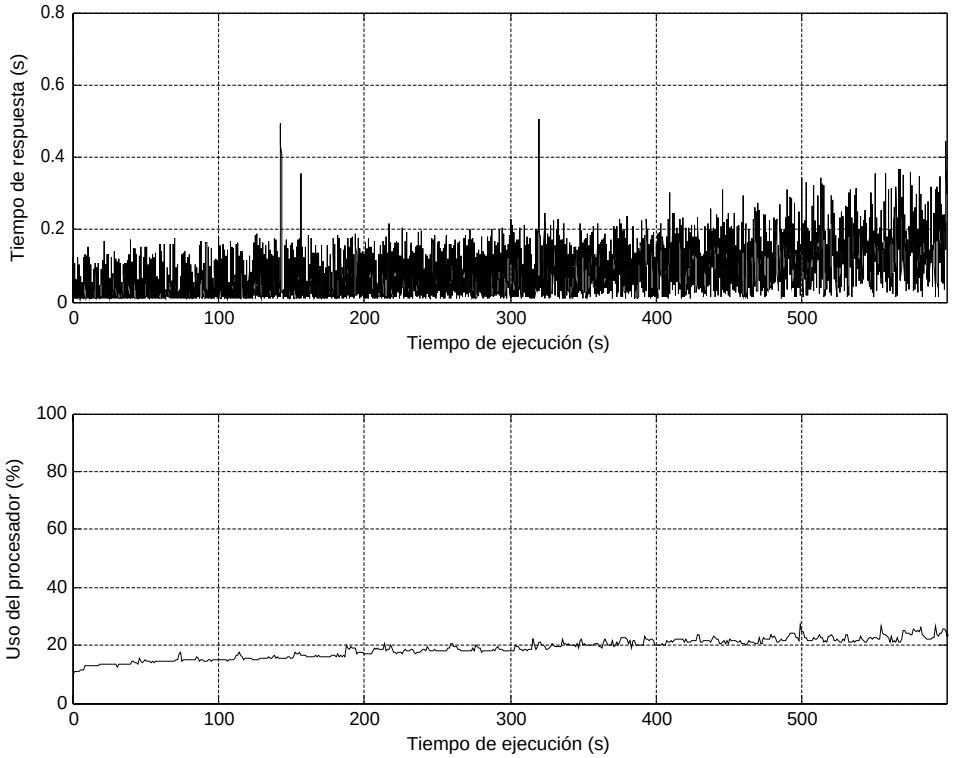

Figura 3.27: Gráficas de la prueba 8 para páginas estáticas en red local.

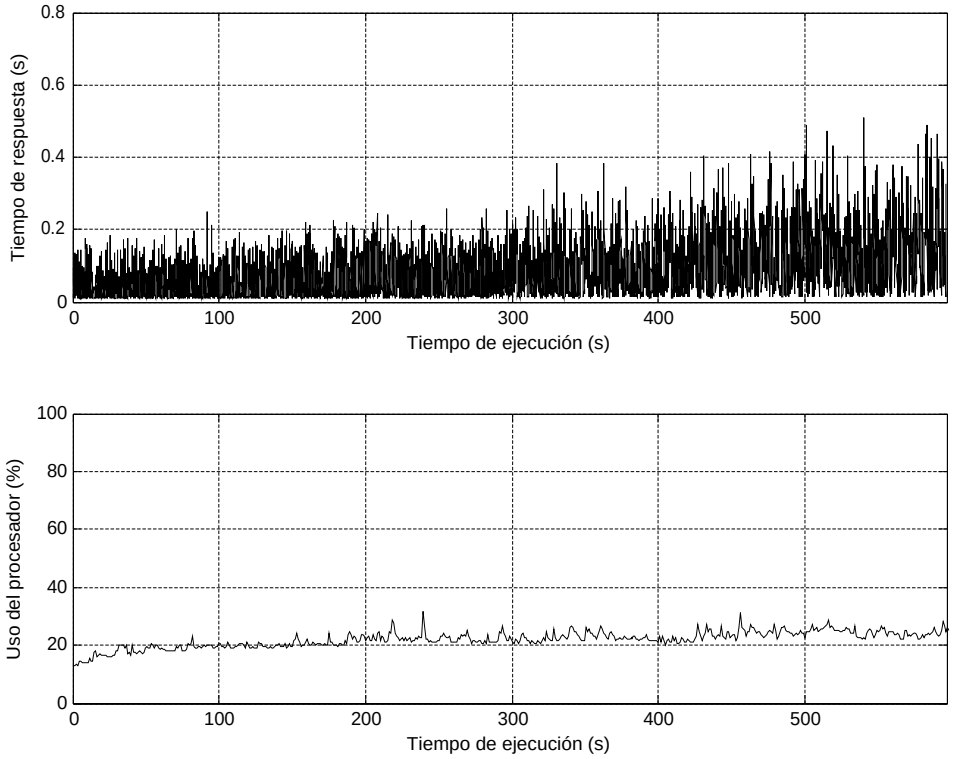

Figura 3.28: Gráficas de la prueba 9 para páginas estáticas en red local.

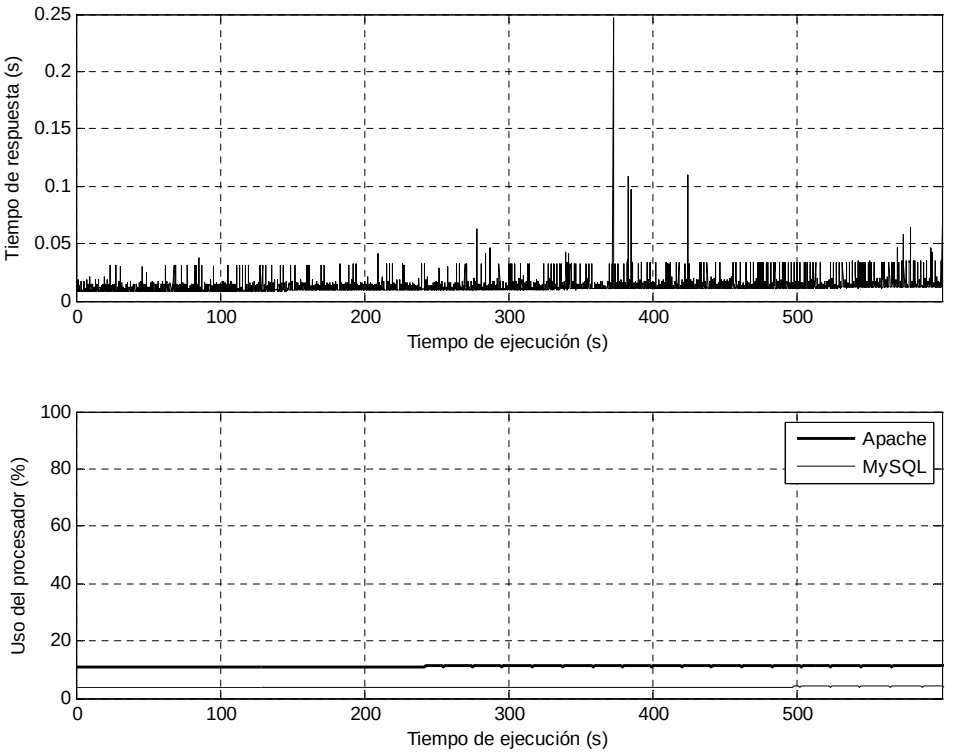

Figura 3.29: Gráficas de la prueba 1 para páginas dinámicas en red local.

3.4.2. Páginas dinámicas

Las gráficas con los resultados de las pruebas se muestran en las figuras 3.29 a 3.37, y las tablas de estadísticas se encuentran en los cuadros 3.30 a 3.38.

Para este conjunto de pruebas se observa que el tiempo de respuesta máximo para el 95 % de las ejecuciones no suele sobrepasar los 150 milisegundos, a pesar de que el tiempo de respuesta medio se mantiene similar al de las pruebas en localhost.

El uso conjunto del procesador del equipo servidor por parte de los procesos de Apache y MySQL se sitúa en torno al 20 % para el 95 % de las ejecuciones realizadas; esto sucede así con las nueve pruebas.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298286	0,000132545
Tiempo real entre peticiones (s)	0,0298286	0,000132246
Tamaño de las páginas (bytes)	6096,51	$1{,}17263 \cdot 10^7$
Número de objetos por página	27,78	260,134
Tiempo de respuesta de páginas (s)	0,0165496	0,000130026

Cuadro 3.31: Estadísticas de la prueba 2 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,03029	0,000874207
Tamaño de las páginas (bytes)	6376,74	$2{,}64078 \cdot 10^7$
Número de objetos por página	28,58	508,738
Tiempo de respuesta de páginas (s)	0,023802	0,000268768

Cuadro 3.32: Estadísticas de la prueba 3 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302893	0,000877961
Tamaño de las páginas (bytes)	602,691	185853
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0146191	0,000187211

Cuadro 3.33: Estadísticas de la prueba 4 para páginas dinámicas en red local.

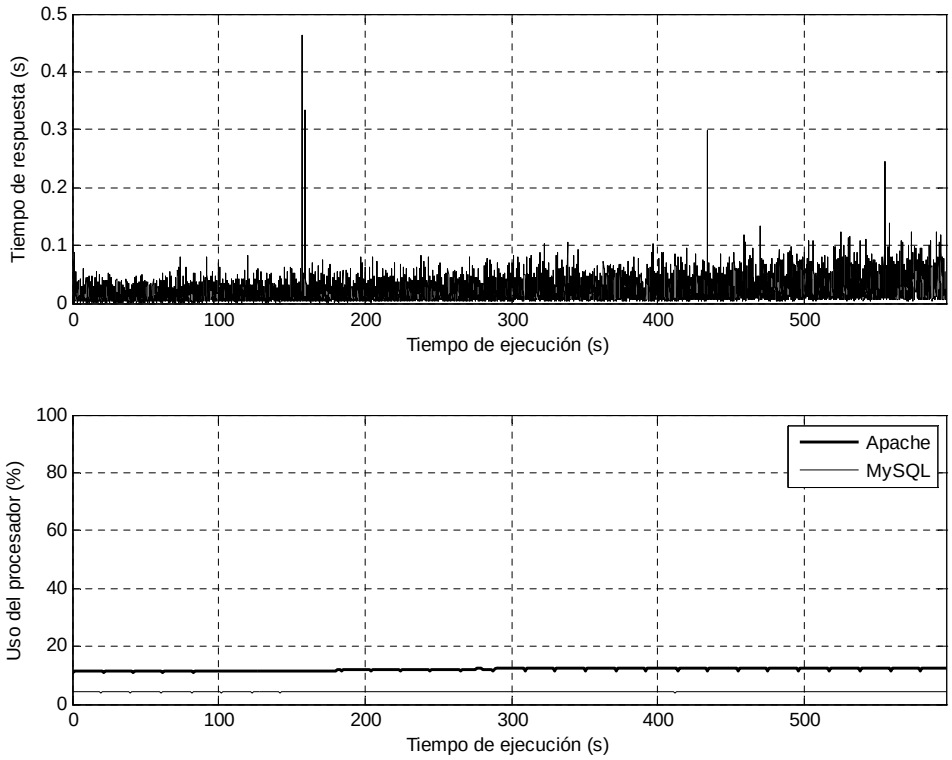

Figura 3.30: Gráficas de la prueba 2 para páginas dinámicas en red local.

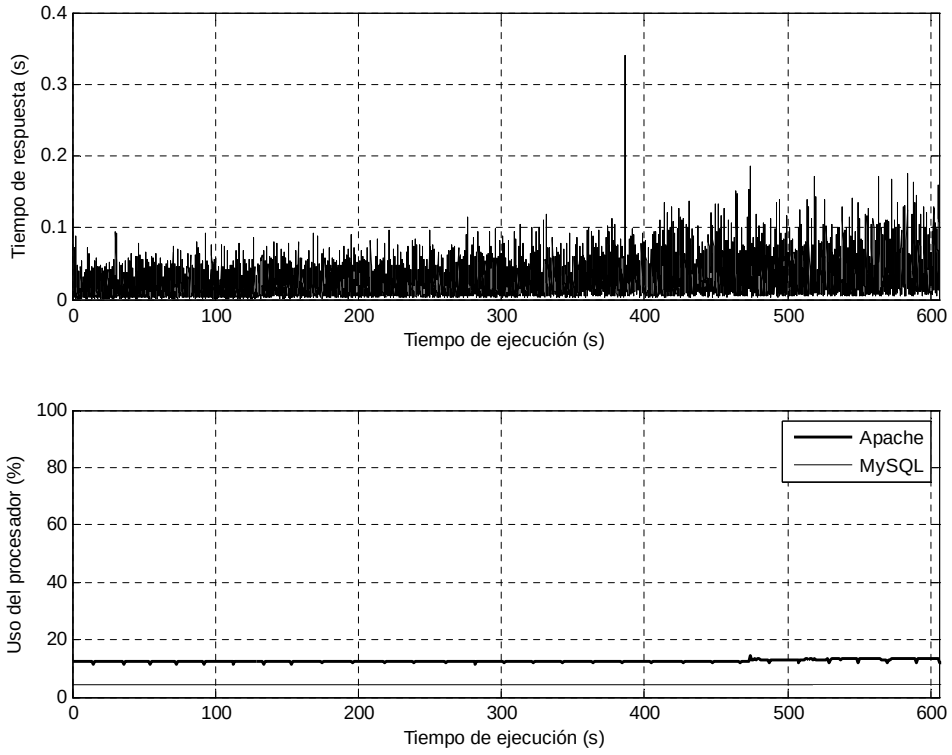

Figura 3.31: Gráficas de la prueba 3 para páginas dinámicas en red local.

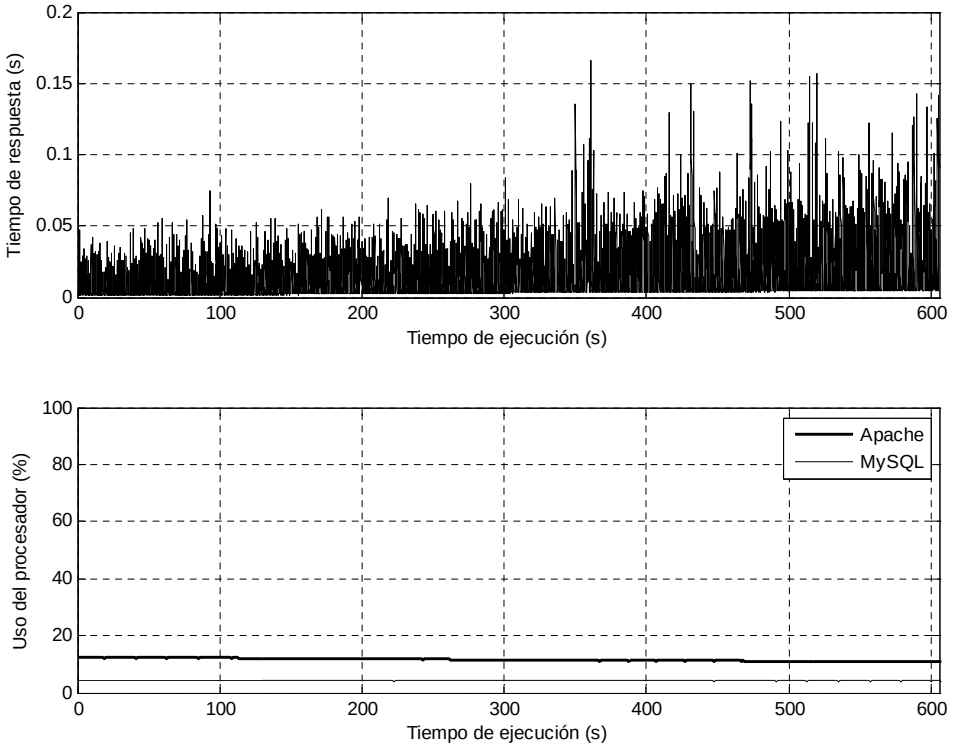

Figura 3.32: Gráficas de la prueba 4 para páginas dinámicas en red local.

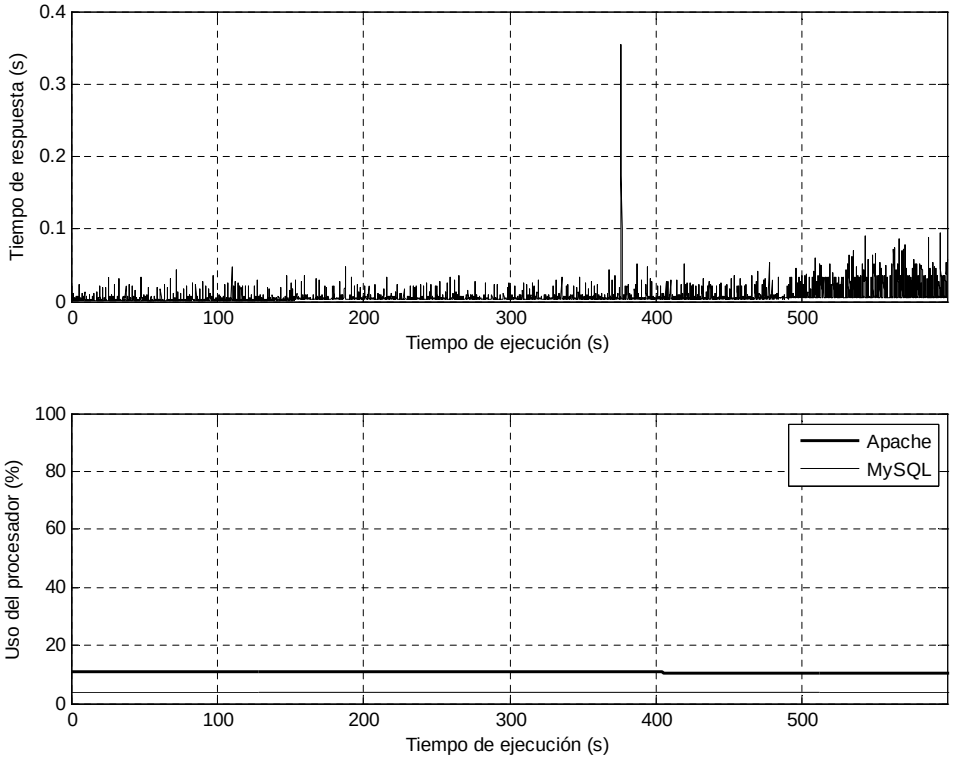

Figura 3.33: Gráficas de la prueba 5 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299418	0,000424774
Tamaño de las páginas (bytes)	602,691	185853
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,00349484	$1{,}0007 \cdot 10^{-5}$

Cuadro 3.34: Estadísticas de la prueba 5 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00156336
Tamaño de las páginas (bytes)	597,91	177798
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,0164595	0,00022277

Cuadro 3.35: Estadísticas de la prueba 6 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0302891	0,000890426
Tiempo real entre peticiones (s)	0,0302895	0,000872859
Tamaño de las páginas (bytes)	6098,45	$5{,}2214 \cdot 10^{6}$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,023757	0,000219621

Cuadro 3.36: Estadísticas de la prueba 7 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0299417	0,000425346
Tiempo real entre peticiones (s)	0,0299418	0,000423934
Tamaño de las páginas (bytes)	6098,45	$5{,}2214 \cdot 10^{6}$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,0143646	$6{,}2914 \cdot 10^{-5}$

Cuadro 3.37: Estadísticas de la prueba 8 para páginas dinámicas en red local.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,0298297	0,00158322
Tiempo real entre peticiones (s)	0,0298297	0,00155635
Tamaño de las páginas (bytes)	6115,84	$5{,}24249 \cdot 10^{6}$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,0261666	0,000257658

Cuadro 3.38: Estadísticas de la prueba 9 para páginas dinámicas en red local.

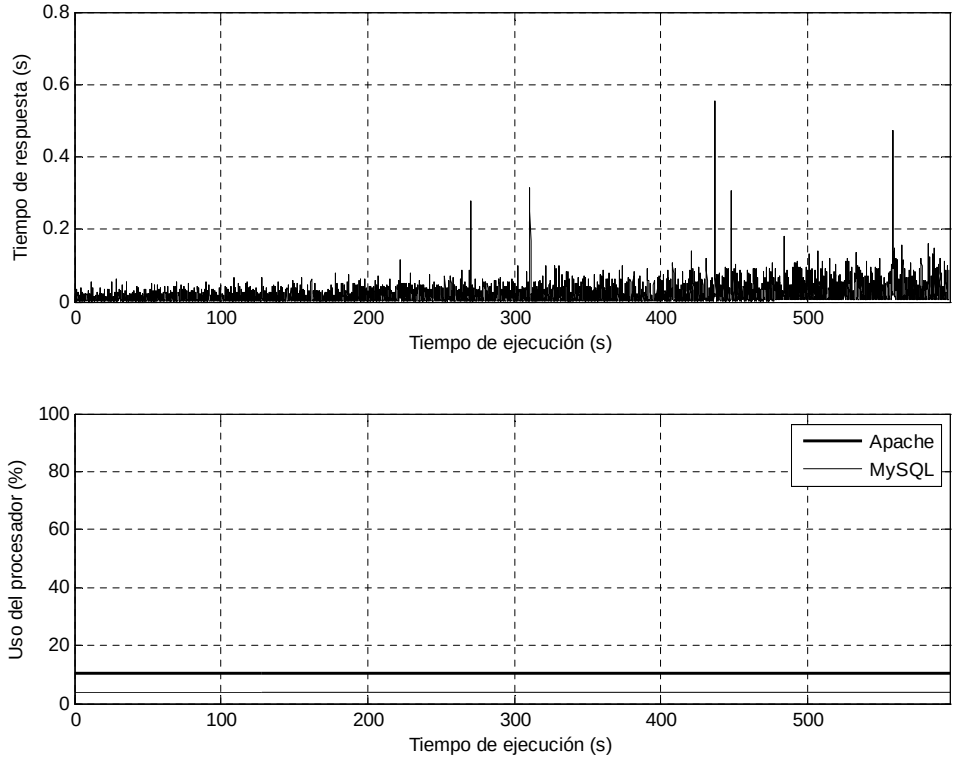

Figura 3.34: Gráficas de la prueba 6 para páginas dinámicas en red local.

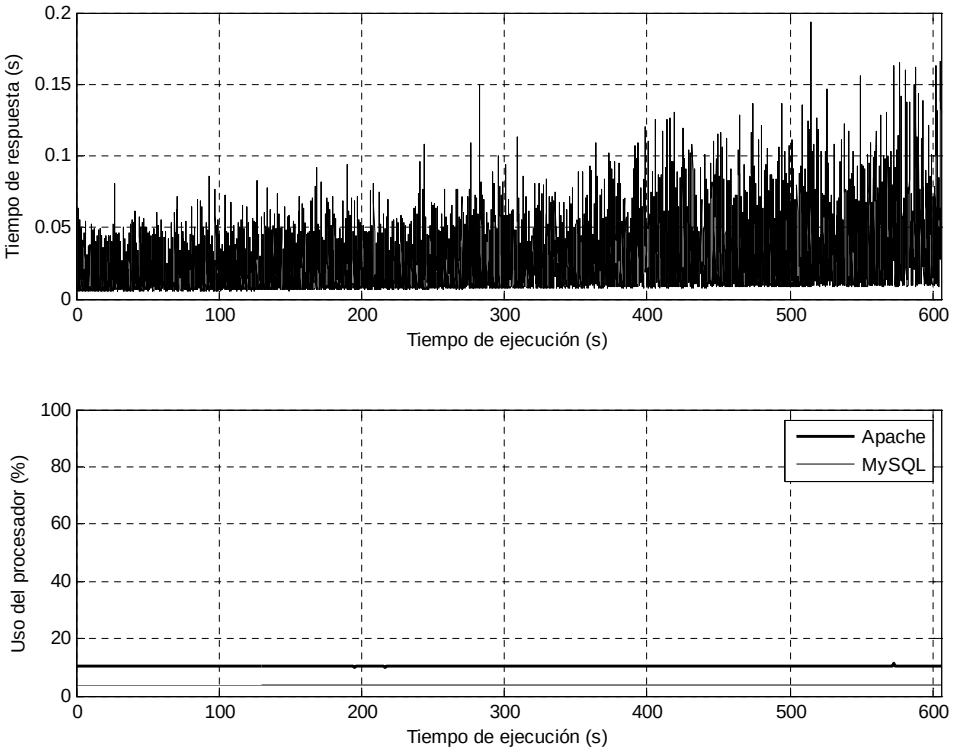

Figura 3.35: Gráficas de la prueba 7 para páginas dinámicas en red local.

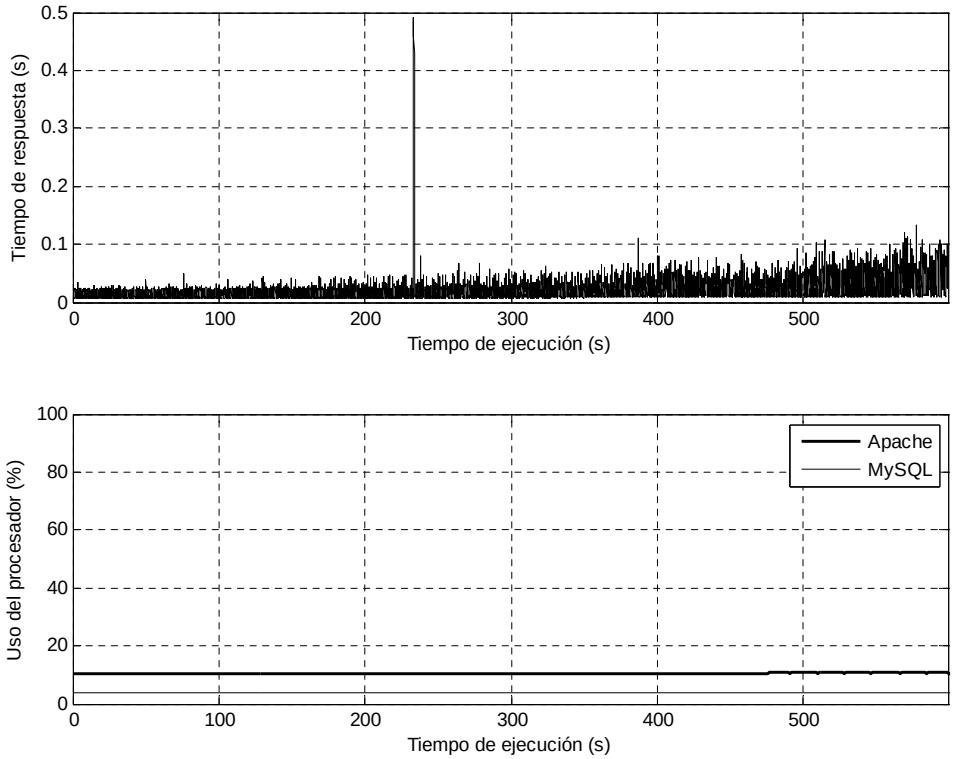

Figura 3.36: Gráficas de la prueba 8 para páginas dinámicas en red local.

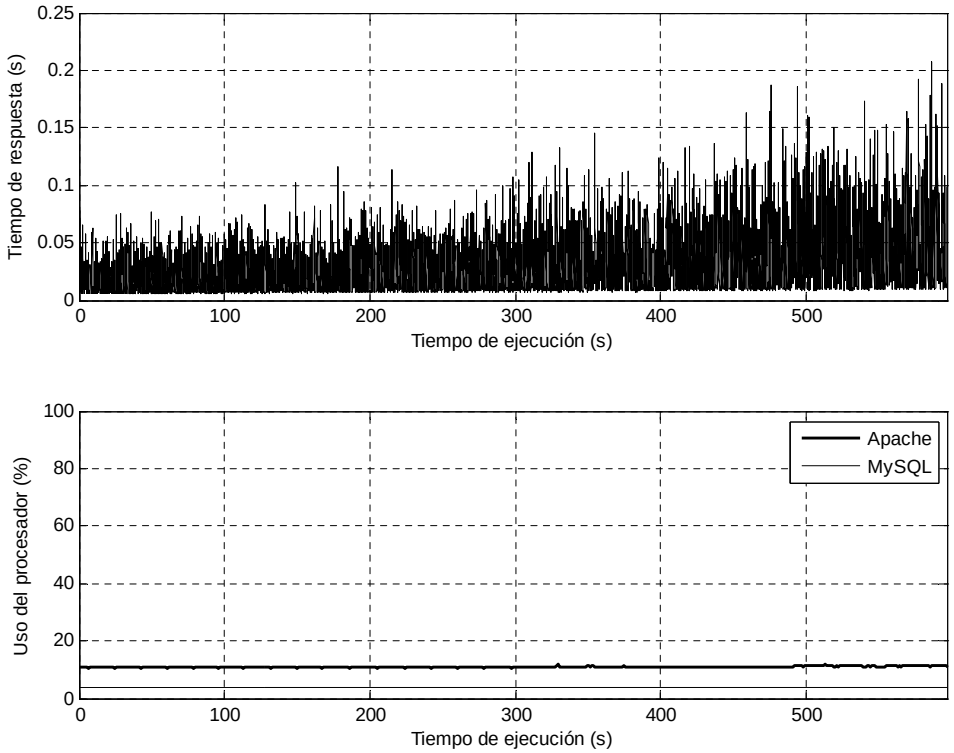

Figura 3.37: Gráficas de la prueba 9 para páginas dinámicas en red local.

3.5. Pruebas en Internet

En las pruebas que se documentan en esta sección, las peticiones del cliente y las respuestas del servidor tienen que viajar a través de Internet para llegar hasta sus destinos. El esquema que se ha utilizado es el siguiente:

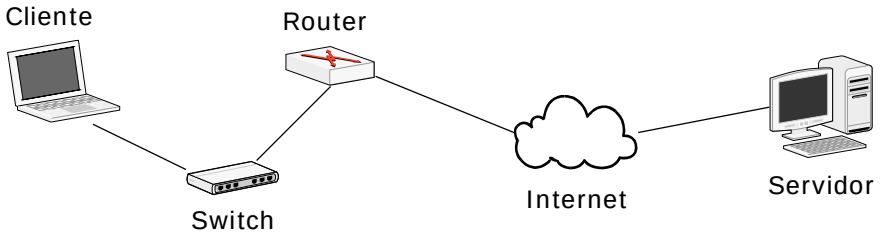

Figura 3.38: Topología usada con las pruebas en Internet.

El equipo cliente es el mismo portátil que se ha usado en las cuatro pruebas anteriores. Dispone de una dirección de red privada y se conecta con un *switch* Fast Ethernet, que a su vez está conectado con un *router* ADSL. De esta forma, el *switch* redirige hacia el *router* todas las peticiones que el cliente realiza al exterior.
El equipo servidor es un ordenador de sobremesa, con un procesador Intel Pentium 4 HT a 3200 MHz y 768 MB de memoria RAM, que se encuentra situado en la Universidad. Durante la ejecución de las pruebas se pudo acceder a él directamente a través de una dirección de red pública.
Una traza realizada al servidor desde el equipo cliente permite conocer la ruta que siguen los paquetes a través de la red:

```
    1  192.168.1.1
    2  192.168.153.1
    3  130.Red-81-46-43.staticIP.rima-tde.net [81.46.43.130]
    4  70.Red-80-58-75.staticIP.rima-tde.net [80.58.75.70]
    5  rediris-2.espanix.net [193.149.1.154]
    6  130.206.206.201
    7  XE4-0-0.EB-IRIS4.red.rediris.es [130.206.250.2]
    8  NAC.XE0-1-0.EB-Valencia0.red.rediris.es [130.206.250.34]
    9  umh-router.red.rediris.es [130.206.211.46]
   10  umh-142-1.umh.es [193.147.142.1]
11-13  no reply
   14  193.147.148.240
```

	Tiempo entre peticiones	Tamaño de objetos	Objetos por página
Prueba 1	Constante $c = 1$	Constante $c = 200$	Constante $c = 30$
Prueba 2	Distr. uniforme $v_{min} = 0{,}5$ $v_{max} = 1{,}5$	Distr. uniforme $v_{min} = 0$ $v_{max} = 400$	Distr. uniforme $v_{min} = 0$ $v_{max} = 60$
Prueba 3	Distr. exponencial $\mu = 1$	Distr. exponencial $\mu = 200$	Distr. exponencial $\mu = 30$
Prueba 4	Distr. exponencial $\mu = 1$	Distr. Pareto $x_m = 57$, $k = 1{,}4$	Constante $c = 1$
Prueba 5	Distr. Pareto $x_m = 0{,}6$, $k = 2{,}5$	Distr. Pareto $x_m = 57$, $k = 1{,}4$	Constante $c = 1$
Prueba 6	Distr. hiperexponencial $\mu_1 = 0{,}5$, $\mu_2 = 2$, $p_1 = 0{,}6667$	Distr. Pareto $x_m = 57$, $k = 1{,}4$	Constante $c = 1$
Prueba 7	Distr. exponencial $\mu = 1$	Distr. lognormal $\mu = 200$, $\sigma^2 = 40000$	Distr. Pareto $x_m = 18$, $k = 2{,}5$
Prueba 8	Distr. Pareto $x_m = 0{,}6$, $k = 2{,}5$	Distr. lognormal $\mu = 200$, $\sigma^2 = 40000$	Distr. Pareto $x_m = 18$, $k = 2{,}5$
Prueba 9	Distr. hiperexponencial $\mu_1 = 0{,}5$, $\mu_2 = 2$, $p_1 = 0{,}6667$	Distr. lognormal $\mu = 200$, $\sigma^2 = 40000$	Distr. Pareto $x_m = 18$, $k = 2{,}5$

Cuadro 3.39: Distribuciones y atributos usados en las pruebas de Internet.

Las distribuciones y parámetros que se han utilizado en cada prueba se muestran en el cuadro 3.39. Los valores de los parámetros se han elegido de tal forma que, para cualquiera de las cinco distribuciones, el valor medio del tiempo entre peticiones es de 1 segundo, el tamaño medio de los objetos es de 200 bytes, y el número medio de objetos por página es 30. En el cuadro 3.40 se resumen las medias y varianzas teóricas de los parámetros seleccionados.

El valor medio del tiempo entre peticiones que se ha seleccionado en este conjunto de pruebas se ha multiplicado por 33 con respecto a los anteriores. Esto se ha hecho así porque el tiempo de respuesta ha aumentado mucho con esta topología de red, y el servidor se saturaba con la anterior frecuencia de envío de peticiones: muchos de los paquetes TCP de respuesta incluían la bandera RST (*reset*) en su cabecera.

El número de peticiones realizadas por cada una de las 90 ejecuciones realizadas por cada tipo de página ha sido de 600. Como el valor medio del tiempo entre peticiones es de un segundo para todas las pruebas, la duración de cada ejecución ha sido de, aproximadamente, diez minutos. El número de ficheros-objeto y de páginas web generadas ha sido de 50 en todas las pruebas. La semilla aleatoria ha sido $s = 241181$.

	Tiempo peticiones	Tamaño objetos	Obj./Pág.
Constante	$\mu = 1$ $\sigma^2 = 0$	$\mu = 200$ $\sigma^2 = 0$	$\mu = 30$ $\sigma^2 = 0$
D. Uniforme	$\mu = 1$ $\sigma^2 = 8{,}333 \cdot 10^{-2}$	$\mu = 200$ $\sigma^2 = 1{,}333 \cdot 10^4$	$\mu = 30$ $\sigma^2 = 300$
D. Exponencial	$\mu = 1$ $\sigma^2 = 1$	$\mu = 200$ $\sigma^2 = 4 \cdot 10^4$	$\mu = 30$ $\sigma^2 = 900$
D. Pareto	$\mu = 1$ $\sigma^2 = 0{,}8$	$\mu \approx 200$ $\sigma^2 = -$	$\mu = 30$ $\sigma^2 = 720$
D. Hiperexpon.	$\mu \approx 1$ $\sigma^2 \approx 2$	—	—
D. Lognormal	—	$\mu = 200$ $\sigma^2 = 4 \cdot 10^4$	—

Cuadro 3.40: Valores medios y varianzas teóricas de las pruebas en Internet.

3.5.1. Páginas estáticas

Las gráficas con los resultados de cada prueba se muestran en las figuras 3.39 a 3.47. No se muestran las gráficas del uso que hace Apache del procesador porque su resultado es constante y muy cercano a cero para todas las ejecuciones. Las tablas con las estadísticas de los resultados se encuentran en los cuadros 3.41 a 3.49.

De acuerdo con las tablas, el tiempo medio de respuesta de las páginas web es muy elevado: supera los 60 segundos en todas las pruebas, excepto en aquéllas en las que las páginas web se componen de un único objeto (pruebas 4, 5 y 6), que se sitúan en torno a los 800 milisegundos.

Además, de acuerdo con las gráficas, se observa que los tiempos máximos de respuesta para el 95 % de las ejecuciones se sitúan en torno a los 200 segundos, alcanzando picos de 400 a 600 segundos en algunos casos. En las pruebas 4, 5 y 6, el tiempo máximo de respuesta es de 4 segundos, con algunos picos de hasta 10 segundos.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1	0
Tiempo real entre peticiones (s)	1	$9{,}94657 \cdot 10^{-8}$
Tamaño de las páginas (bytes)	1704,62	48,4504
Tamaño de los objetos (bytes)	200	0
Número de objetos por página	30	0
Tiempo de respuesta de páginas (s)	72,2625	78,8411
Tiempo de respuesta de objetos (s)	1,96687	1,06724

Cuadro 3.41: Estadísticas de la prueba 1 para páginas estáticas en Internet.

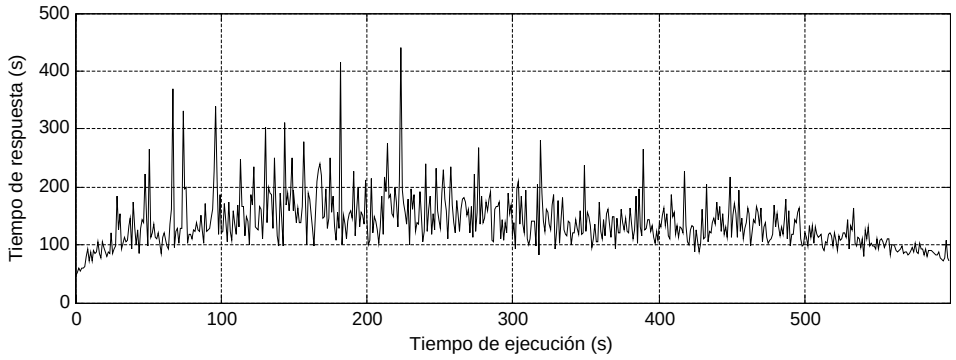

Figura 3.39: Gráficas de la prueba 1 para páginas estáticas en Internet.

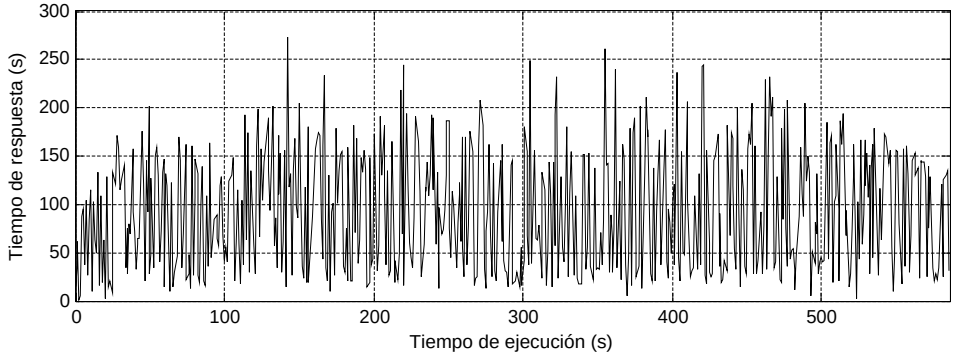

Figura 3.40: Gráficas de la prueba 2 para páginas estáticas en Internet.

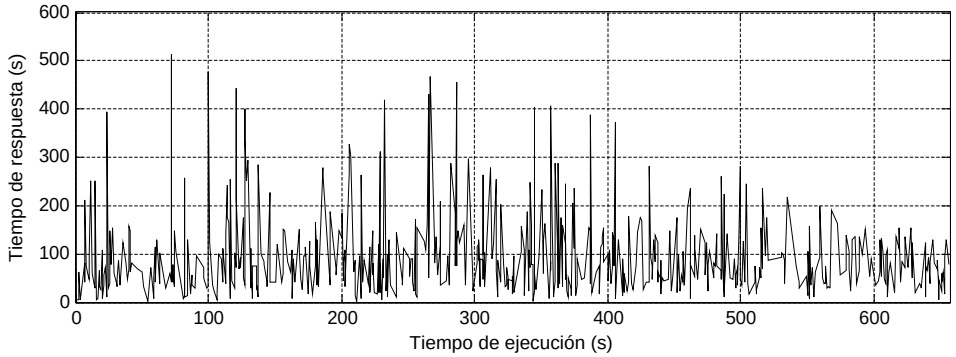

Figura 3.41: Gráficas de la prueba 3 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,978249	0,0831777
Tiempo real entre peticiones (s)	0,978249	0,0831779
Tamaño de las páginas (bytes)	1585,66	437735
Tamaño de los objetos (bytes)	188,64	11577,9
Número de objetos por página	25,98	345
Tiempo de respuesta de páginas (s)	60,884	1670,49
Tiempo de respuesta de objetos (s)	2,00816	0,477714

Cuadro 3.42: Estadísticas de la prueba 2 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,09719	1,26474
Tiempo real entre peticiones (s)	1,09719	1,26475
Tamaño de las páginas (bytes)	1582,96	902741
Tamaño de los objetos (bytes)	192,96	22527,5
Número de objetos por página	26,58	703,596
Tiempo de respuesta de páginas (s)	63,2055	3537,54
Tiempo de respuesta de objetos (s)	2,03368	0,645654

Cuadro 3.43: Estadísticas de la prueba 3 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,09719	1,26474
Tiempo real entre peticiones (s)	1,09719	1,26474
Tamaño de las páginas (bytes)	511,9	$3{,}23657 \cdot 10^{-27}$
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,828689	0,0823061
Tiempo de respuesta de objetos (s)	0,52025	0,0549441

Cuadro 3.44: Estadísticas de la prueba 4 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,08548	1,52925
Tiempo real entre peticiones (s)	1,08548	1,52925
Tamaño de las páginas (bytes)	511,7	$3{,}30162 \cdot 10^{-23}$
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,774243	0,0405235
Tiempo de respuesta de objetos (s)	0,381581	0,00888536

Cuadro 3.45: Estadísticas de la prueba 5 para páginas estáticas en Internet.

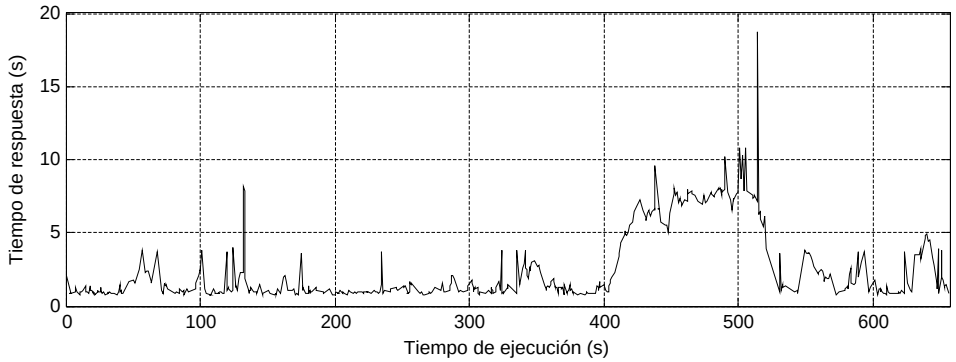

Figura 3.42: Gráficas de la prueba 4 para páginas estáticas en Internet.

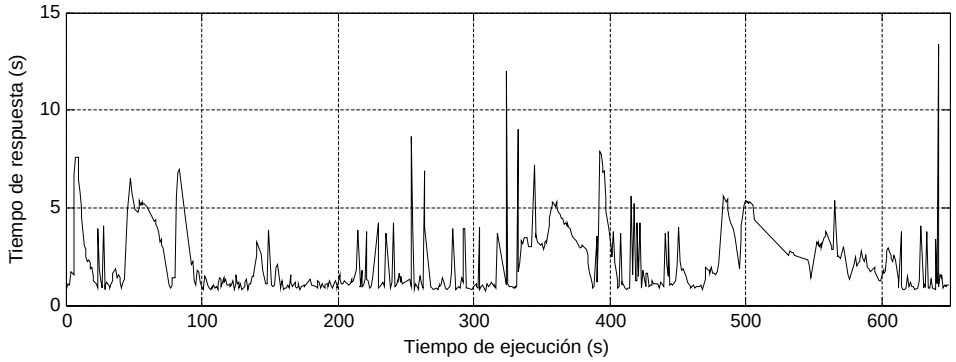

Figura 3.43: Gráficas de la prueba 5 para páginas estáticas en Internet.

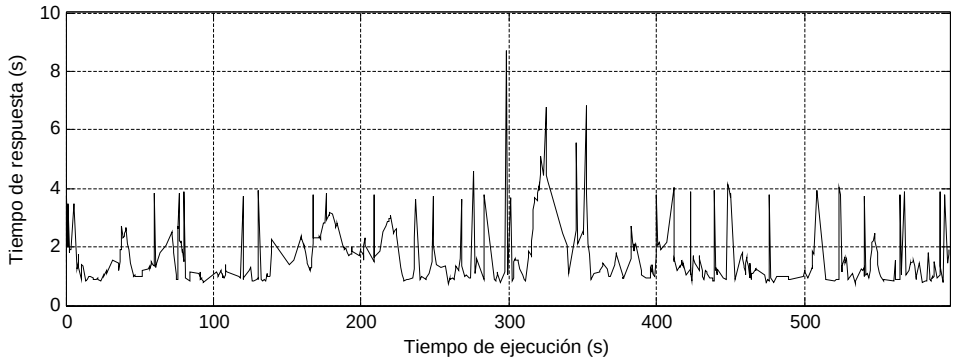

Figura 3.44: Gráficas de la prueba 6 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,99908	1,99775
Tiempo real entre peticiones (s)	0,99908	1,99794
Tamaño de las páginas (bytes)	511,9	$3{,}23657 \cdot 10^{-27}$
Tamaño de los objetos (bytes)	133,96	10167
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,765672	0,0222825
Tiempo de respuesta de objetos (s)	0,373925	0,00435403

Cuadro 3.46: Estadísticas de la prueba 6 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,09719	1,26474
Tiempo real entre peticiones (s)	1,09719	1,26469
Tamaño de las páginas (bytes)	1682,38	437352
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	62,7285	1348,33
Tiempo de respuesta de objetos (s)	1,87578	0,703799

Cuadro 3.47: Estadísticas de la prueba 7 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,08548	1,52925
Tiempo real entre peticiones (s)	1,08548	1,52925
Tamaño de las páginas (bytes)	1682,24	437500
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	64,7652	1407,3
Tiempo de respuesta de objetos (s)	1,88665	0,771883

Cuadro 3.48: Estadísticas de la prueba 8 para páginas estáticas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,99908	1,99775
Tiempo real entre peticiones (s)	0,99908	1,99812
Tamaño de las páginas (bytes)	1696,57	427023
Tamaño de los objetos (bytes)	192,12	44645,3
Número de objetos por página	29,74	350,482
Tiempo de respuesta de páginas (s)	68,2094	1372,9
Tiempo de respuesta de objetos (s)	2,20915	2,00514

Cuadro 3.49: Estadísticas de la prueba 9 para páginas estáticas en Internet.

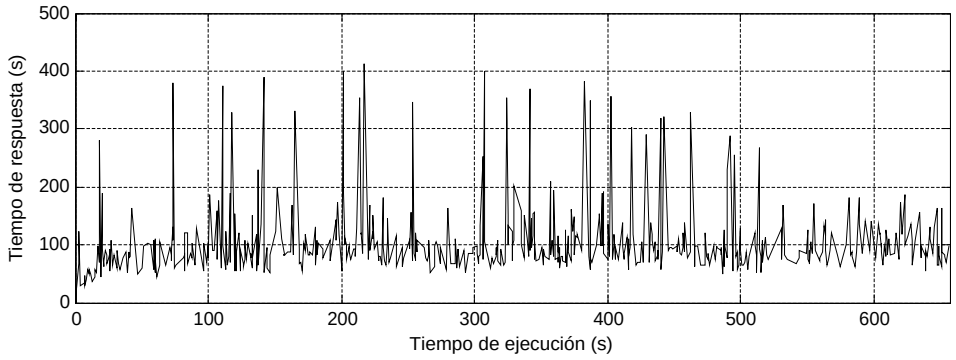

Figura 3.45: Gráficas de la prueba 7 para páginas estáticas en Internet.

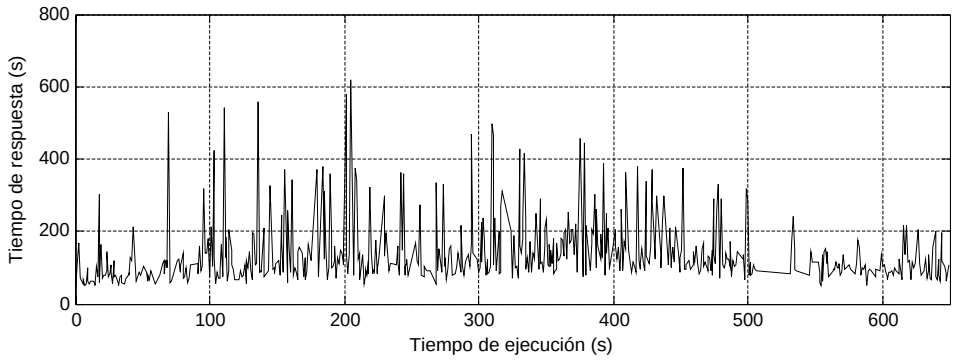

Figura 3.46: Gráficas de la prueba 8 para páginas estáticas en Internet.

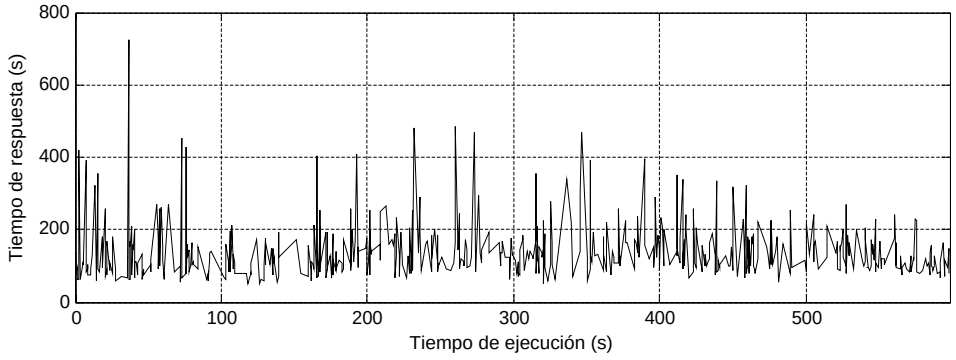

Figura 3.47: Gráficas de la prueba 9 para páginas estáticas en Internet.

3.5.2. Páginas dinámicas

Las gráficas con los resultados de cada prueba se muestran en las figuras 3.48 a 3.56. Igual que en las pruebas con páginas web estáticas, no se muestran las gráficas del uso del procesador porque su resultado también es constante y muy cercano a cero para todas las ejecuciones, tanto para Apache como para MySQL. Las tablas con las estadísticas de los resultados se encuentran en los cuadros 3.50 a 3.58.

Observando las tablas se comprueba que el tiempo medio de respuesta de las páginas dinámicas es bastante inferior al de las páginas estáticas: en ninguna prueba se superan los 400 milisegundos. Es decir, el tiempo medio de respuesta es inferior al tiempo medio de espera entre peticiones, que es de un segundo. Esto ocurre con las páginas dinámicas porque el cliente sólo necesita realizar una petición para descargarlas por completo.

De acuerdo con las gráficas, los tiempos máximos de respuesta para el 95 % de las ejecuciones muestran algunos picos aislados de entre 3 y 10 segundos en todas las pruebas.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1	0
Tiempo real entre peticiones (s)	1	$1{,}80683 \cdot 10^{-6}$
Tamaño de las páginas (bytes)	6626	0
Número de objetos por página	30	0
Tiempo de respuesta de páginas (s)	0,382684	0,00344051

Cuadro 3.50: Estadísticas de la prueba 1 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,978249	0,0831777
Tiempo real entre peticiones (s)	0,978249	0,0831784
Tamaño de las páginas (bytes)	5977,52	$1{,}15744 \cdot 10^{7}$
Número de objetos por página	27,78	260,134
Tiempo de respuesta de páginas (s)	0,389896	0,0168192

Cuadro 3.51: Estadísticas de la prueba 2 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,09719	1,26474
Tiempo real entre peticiones (s)	1,09719	1,26474
Tamaño de las páginas (bytes)	6681,05	$2{,}94469 \cdot 10^{7}$
Número de objetos por página	28,58	508,738
Tiempo de respuesta de páginas (s)	0,393169	0,0198966

Cuadro 3.52: Estadísticas de la prueba 3 para páginas dinámicas en Internet.

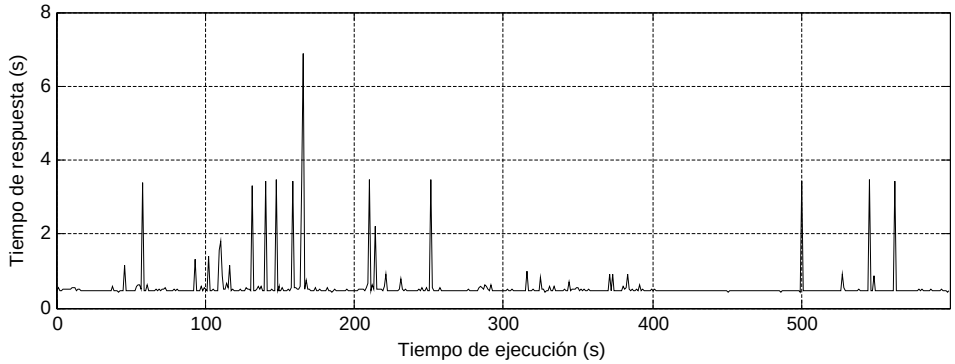

Figura 3.48: Gráfica de la prueba 1 para páginas dinámicas en Internet.

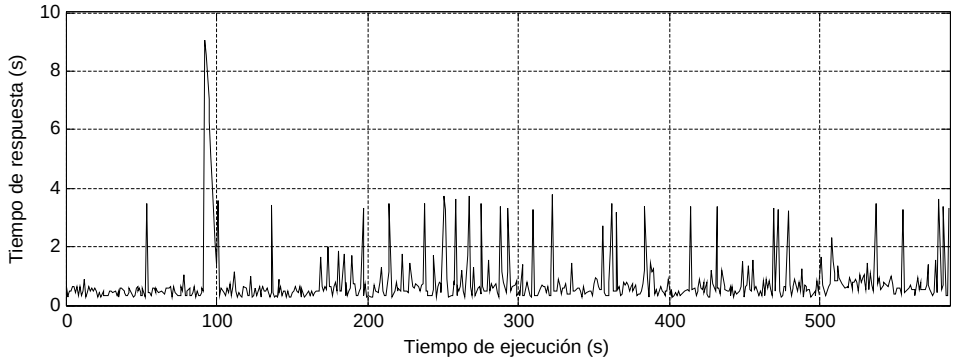

Figura 3.49: Gráfica de la prueba 2 para páginas dinámicas en Internet.

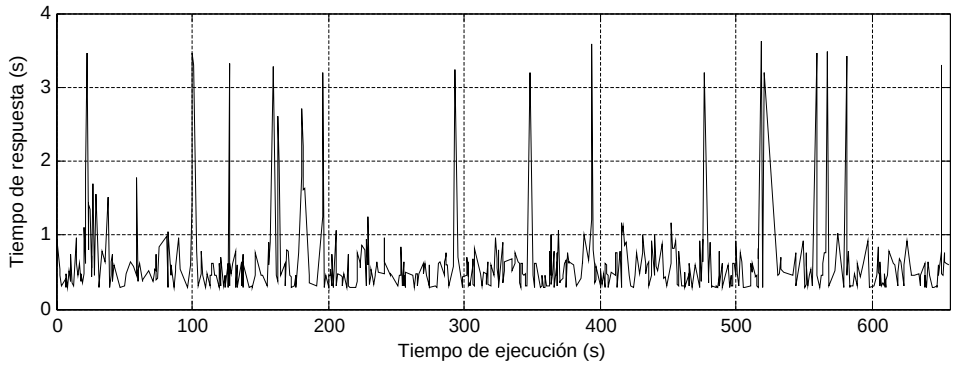

Figura 3.50: Gráfica de la prueba 3 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,09719	1,26474
Tiempo real entre peticiones (s)	1,09719	1,26474
Tamaño de las páginas (bytes)	599,34	174932
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,251125	0,00313891

Cuadro 3.53: Estadísticas de la prueba 4 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,08548	1,52925
Tiempo real entre peticiones (s)	1,08548	1,52924
Tamaño de las páginas (bytes)	599,34	174932
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,253452	0,00217598

Cuadro 3.54: Estadísticas de la prueba 5 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,99908	1,99775
Tiempo real entre peticiones (s)	0,99908	1,99774
Tamaño de las páginas (bytes)	579,207	127555
Número de objetos por página	1	0
Tiempo de respuesta de páginas (s)	0,268644	0,00655813

Cuadro 3.55: Estadísticas de la prueba 6 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,09719	1,26474
Tiempo real entre peticiones (s)	1,09719	1,26474
Tamaño de las páginas (bytes)	6202,49	$5{,}85415 \cdot 10^6$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,391448	0,0139336

Cuadro 3.56: Estadísticas de la prueba 7 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	1,08548	1,52925
Tiempo real entre peticiones (s)	1,08548	1,52925
Tamaño de las páginas (bytes)	6202,49	$5{,}85415 \cdot 10^6$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,395993	0,0180358

Cuadro 3.57: Estadísticas de la prueba 8 para páginas dinámicas en Internet.

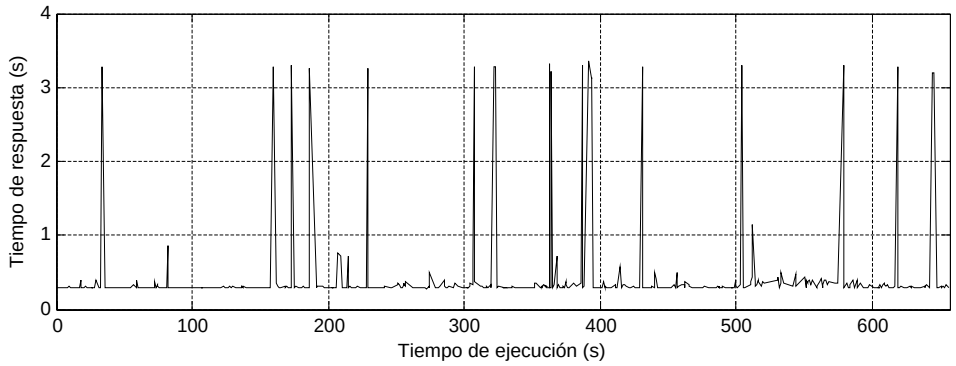

Figura 3.51: Gráfica de la prueba 4 para páginas dinámicas en Internet.

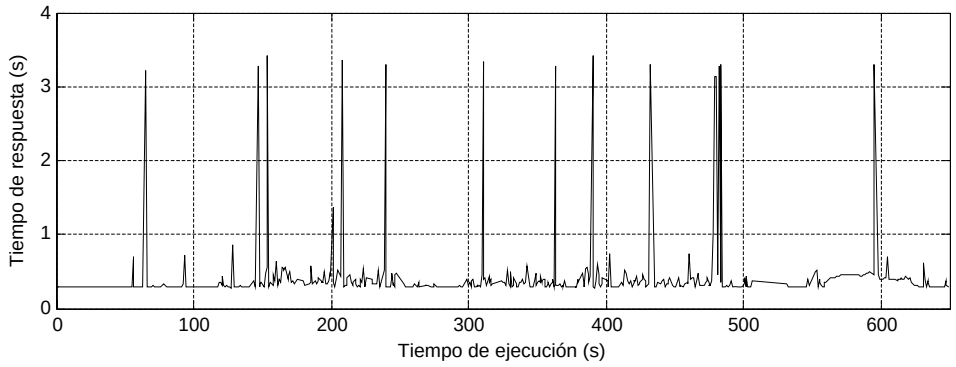

Figura 3.52: Gráfica de la prueba 5 para páginas dinámicas en Internet.

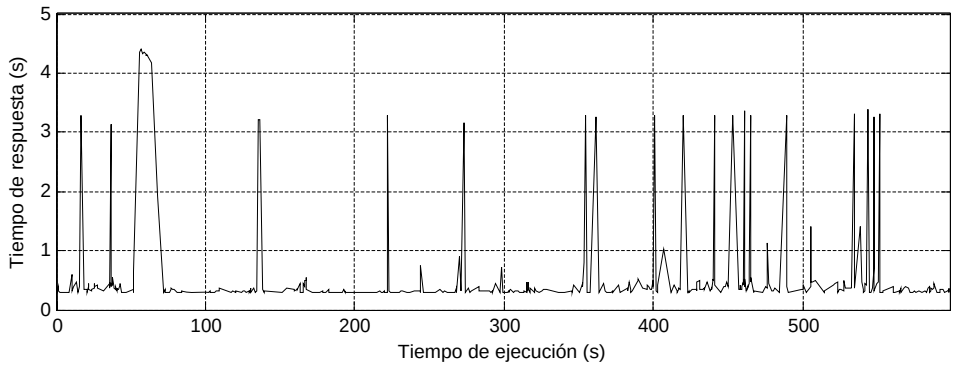

Figura 3.53: Gráfica de la prueba 6 para páginas dinámicas en Internet.

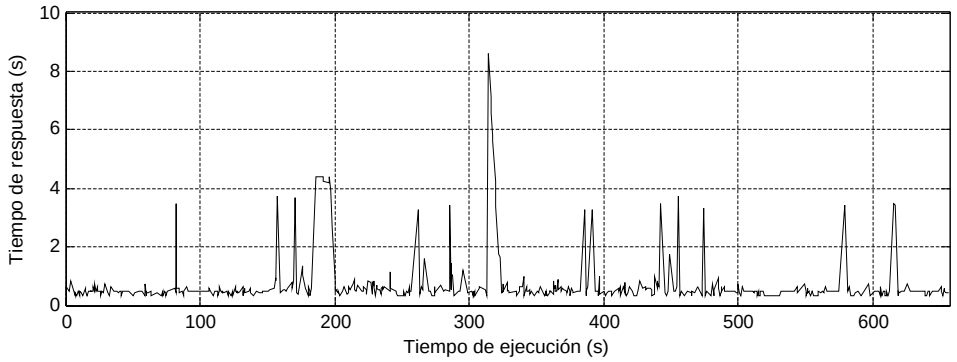

Figura 3.54: Gráfica de la prueba 7 para páginas dinámicas en Internet.

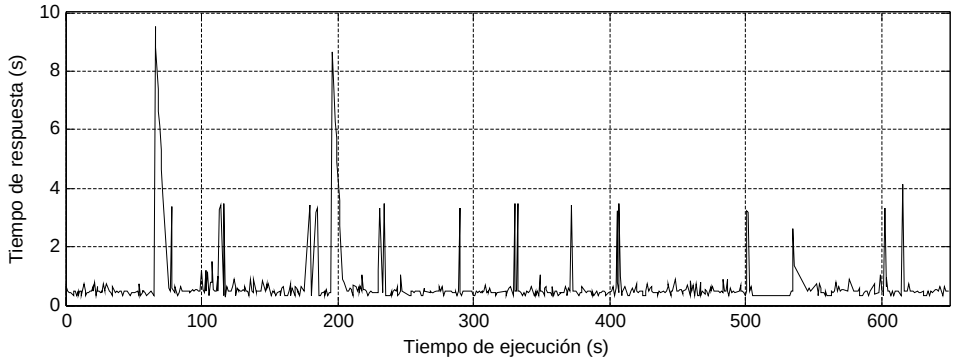

Figura 3.55: Gráfica de la prueba 8 para páginas dinámicas en Internet.

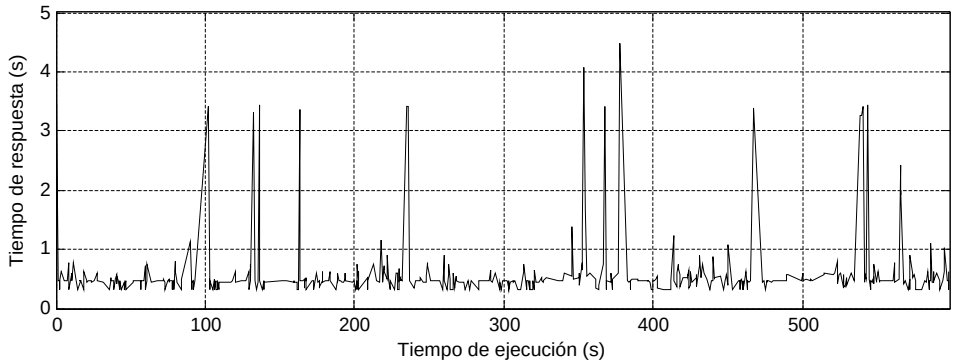

Figura 3.56: Gráfica de la prueba 9 para páginas dinámicas en Internet.

	Media	Varianza
Tiempo teórico entre peticiones (s)	0,99908	1,99775
Tiempo real entre peticiones (s)	0,99908	1,99776
Tamaño de las páginas (bytes)	6120,62	$5{,}02269 \cdot 10^6$
Número de objetos por página	27,4	99,1429
Tiempo de respuesta de páginas (s)	0,370799	0,0076717

Cuadro 3.58: Estadísticas de la prueba 9 para páginas dinámicas en Internet.

3.6. Conclusiones

A partir de los resultados obtenidos se pueden extraer algunas conclusiones generales. Primero, a mayor distancia entre los equipos cliente y servidor, mayor es el tiempo medio de respuesta. Este hecho está relacionado con el número de nodos que deben atravesar los paquetes antes de llegar a su destino, y se ve más claramente con las páginas estáticas, que tienen que realizar varias peticiones para descargarse de forma completa. A continuación, se muestra una tabla comparativa con algunas pruebas y sus tiempos de respuesta para diferentes topologías de red:

	Localhost	Red local	Internet
Prueba 1 (s)	0,0178	0,0385	72,2625
Prueba 4 (s)	0,0092	0,0155	0,8287
Prueba 7 (s)	0,0305	0,0663	62,7285

Cuadro 3.59: Comparación de los tiempos de respuesta de algunas pruebas.

Se aprecia cómo los menores tiempos de respuesta se obtienen en localhost y los mayores en Interet. Esto, además, sin tener en cuenta que las características de los equipos que se han usado como servidores han sido diferentes en cada topología.

Por otro lado, cuanto mayor es el número medio de peticiones que se realizan al servidor en un segundo, más recursos consume su procesador. En las pruebas en localhost y red local, donde se realiza una media de 33 peticiones por segundo, los procesos de Apache y MySQL ocupan entre un 15 % y un 30 % del procesador; pero en las pruebas en Internet, donde la media está en una petición por segundo, el uso del procesador es muy cercano a cero.

Por último, las pruebas cuyas páginas web tienen menor número de objetos ofrecen menores tiempos de respuesta y menores porcentajes de uso del procesador. Esto se puede comprobar fácilmente si se comparan las pruebas 4, 5 y 6 de cualquier topología de red y para cualquier tipo de página web, en las que se trabaja con un único objeto,

con el resto, que trabaja con una media de 30 objetos.

Capítulo 4

VALIDACIÓN

Para validar el funcionamiento del benchmark se van a comparar los valores que se han generado en las pruebas del capítulo anterior con los valores que se deberían haber obtenido según la teoría.
En la sección 4.1 se enumeran los programas de Matlab que se han desarrollado para analizar los datos y representarlos mediante histogramas. En la sección 4.2 se exponen los histogramas según su distribución de probabilidad, y se argumenta si se corresponden con los valores teóricos. Por último, en la sección 4.3, se exponen las conclusiones de los análisis.

4.1. Programas para la validación de los resultados

Se han desarrollado seis programas en Matlab para representar funciones e histogramas a partir de los ficheros generados por el benchmark. Los gráficos obtenidos, que se muestran en las siguientes secciones, se emplean para analizar el funcionamiento del benchmark. A continuación se comenta la utilidad de cada uno de los programas:

- `resta.m`. Resta cada elemento de un vector con su elemento inmediatamente anterior. Esta operación es útil para obtener los tiempos de espera entre peticiones a partir de los ficheros `csv`, ya que éstos sólo guardan el número de microsegundos transcurrido desde que se inicia la prueba hasta que se realiza cada petición (es decir, un vector de valores relativos y crecientes). El vector resultante tiene un elemento menos que el vector original.

- `tomaN.m`. Lee de un fichero `csv` la variable `tiempoInicioPag` y la convierte en un vector de Matlab.

- `tomaNbig.m`. Lee de los diez ficheros `csv` generados por una misma prueba la variable `tiempoInicioPag` y la convierte en un vector de Matlab.
- `tomaO.m`. Lee de un fichero `csv` la variable `tamanoObjetosPagina` y la convierte en un vector de Matlab.
- `tomaP.m`. Lee de un fichero `csv` la variable `objetosPagina` y la convierte en un vector de Matlab (las tres variables extraídas por los cuatro últimos programas se comentaron en la sección 2.6.9).
- `qval.c`. Representa los valores de un vector en un histograma (con un número de clases definido por el usuario y de igual amplitud) cuya área es igual a 1. Además, el programa puede superponer en la gráfica del histograma la curva teórica de la función de probabilidad que se desee. Esto es útil para comparar cuánto se parece el valor teórico al histograma resultante.

El código M de los seis programas se puede encontrar en el DVD adjunto.

4.2. Análisis de resultados

Para analizar los resultados del capítulo anterior se han agrupado las distribuciones de probabilidad con parámetros iguales (a partir de las tablas 3.1 y 3.39), y sus muestras se han representado de forma conjunta en un mismo histograma. Los 20 grupos de distribuciones que se pueden formar se detallan en la tabla 4.1.
Los grupos más importantes son los que representan el tiempo de espera entre peticiones (marcados en la tabla con la etiqueta TEP), ya que sus histogramas permiten conocer si el algoritmo de temporización usado por el benchmark (explicado en la sección 2.6.7) funciona correctamente. El número de muestras disponibles de este tipo es muy elevado, por lo que los histogramas generados son bastante fiables.
Para representar el número de objetos por página y el tamaño de los objetos sólo se dispone de un conjunto de 50 muestras por cada grupo, ya que se ha utilizado la misma semilla para generar los números aleatorios. El número de muestras no es muy representativo, sobre todo si se compara con las decenas de miles (en algunos casos millones) que se emplean para evaluar el tiempo de espera entre peticiones. Sin embargo, la tendencia de los histogramas resultantes también es útil para validar las distribuciones.
En las secciones siguientes se analizan, ordenados por distribuciones de probabilidad, los grupos 1 a 17 de la tabla 4.1. Los grupos 18 a 20, en los que se opera con valores puramente constantes, se han obviado por su poca relevancia.

n.º	Distribución	Parámetros	Opc.*	Id. prueba	Topologías†
1	Uniforme	$v_{min} = 1 \cdot 10^4$, $v_{max} = 5 \cdot 10^4$	TEP	2	LH, RL
2	Uniforme	$v_{min} = 0{,}5 \cdot 10^6$, $v_{max} = 1{,}5 \cdot 10^6$	TEP	2	IN
3	Uniforme	$v_{min} = 0$, $v_{max} = 400$	TDO	2	LH, RL, IN
4	Uniforme	$v_{min} = 0$, $v_{max} = 60$	OPP	2	LH, RL, IN
5	Exponencial	$\mu = 3 \cdot 10^4$	TEP	3, 4, 7	LH, RL
6	Exponencial	$\mu = 1 \cdot 10^6$	TEP	3, 4, 7	IN
7	Exponencial	$\mu = 200$	TDO	3	LH, RL, IN
8	Exponencial	$\mu = 30$	OPP	3	LH, RL, IN
9	Hiperexpon.	$\mu_1 = 25 \cdot 10^3$, $\mu_2 = 1 \cdot 10^5$, $p_1 = 0{,}9333$	TEP	6, 9	LH, RL
10	Hiperexpon.	$\mu_1 = 0{,}5 \cdot 10^6$, $\mu_2 = 2 \cdot 10^6$, $p_1 = 0{,}6667$	TEP	6, 9	IN
11	Pareto	$x_m = 18 \cdot 10^3$, $k = 2{,}5$	TEP	5, 8	LH, RL
12	Pareto	$x_m = 6 \cdot 10^5$, $k = 2{,}5$	TEP	5, 8	IN
13	Pareto	$x_m = 57$, $k = 1{,}4$	TDO	4, 5, 6	LH, RL, IN
14	Pareto	$x_m = 18$, $k = 2{,}5$	OPP	7, 8, 9	LH, RL, IN
15	Lognormal	$\mu = 200$, $\sigma^2 = 40\,000$	TDO	7, 8, 9	LH, RL, IN
16	Constante	$c = 3 \cdot 10^4$	TEP	1	LH, RL
17	Constante	$c = 1 \cdot 10^6$	TEP	1	IN
18	Constante	$c = 200$	TDO	1	LH, RL, IN
19	Constante	$c = 30$	OPP	1	LH, RL, IN
20	Constante	$c = 1$	OPP	4, 5, 6	LH, RL, IN

* Opción sobre la que se aplica la distribución: TEP = tiempo de espera entre peticiones; TDO = tamaño de los objetos; OPP = número de objetos por página.

† Topología sobre la que se aplica la distribución: LH = localhost; RL = red local; IN = Internet.

Cuadro 4.1: Grupos de distribuciones realizados para analizar los resultados.

4.2.1. Distribución uniforme

Grupo 1

Se analizan los datos generados a partir de una distribución uniforme con $v_{min} = 1 \cdot 10^4$ μs y $v_{max} = 5 \cdot 10^4$ μs. Estas características se cumplen para el tiempo de espera entre peticiones de la prueba 2, con páginas web estáticas y dinámicas, y en localhost y red local.

En total se obtienen 799 960 muestras con las que se ha representado el histograma de 100 clases que se muestra en la figura 4.1.

En la misma figura se ha superpuesto la función teórica con líneas de puntos. Esta función, obtenida a partir de la ecuación 2.2, es la siguiente:

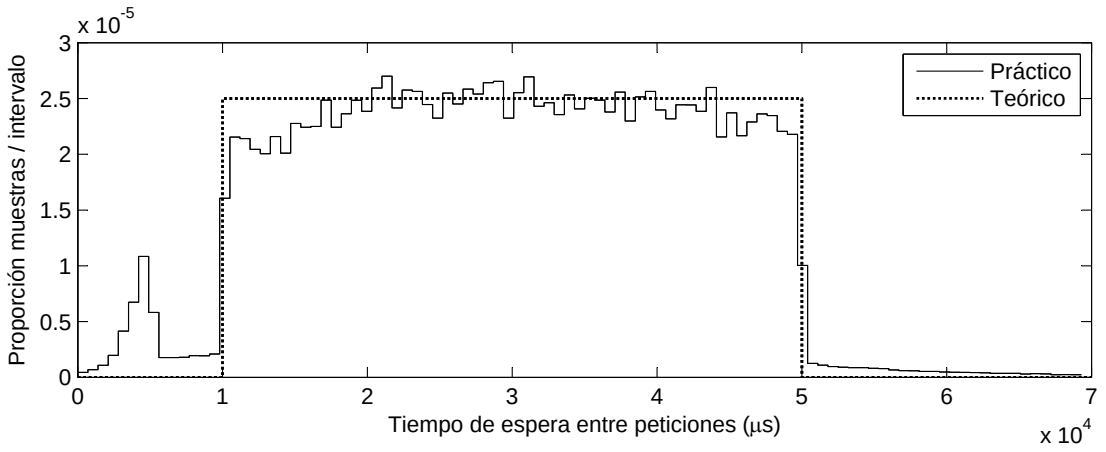

Figura 4.1: Histograma para validar el grupo de distribuciones n.º 1.

$$f(x) = \begin{cases} 2{,}5 \cdot 10^{-5} & \text{para } 1 \cdot 10^4 \leq x \leq 5 \cdot 10^4, \\ 0 & \text{para cualquier otro punto} \end{cases} \tag{4.1}$$

Se observa cómo el histograma se ajusta bastante bien al resultado teórico salvo por tres aspectos: el pico que aparece en el intervalo comprendido entre los 0 y 10 ms, que no debería aparecer; el intervalo de los 10 a los 20 ms en el que las muestras se encuentran ligeramente por debajo de su valor teórico; y la pequeña cola del margen izquierdo, que tampoco debería aparecer.

Como se explicó en la sección 2.6.7, el tiempo de espera entre peticiones puede sufrir algunos retrasos, que son más frecuentes cuando los intervalos son muy pequeños (del orden de centésimas de segundo, como en este caso). El hecho de que una petición se realice con retraso no altera al resto de peticiones, que se realizan en los instantes previstos desde un principio.

Por esta razón, si el tiempo de espera de una petición se alarga por un retraso, el tiempo de espera de la siguiente petición se tiene que acortar para cumplir con los plazos, aunque esto implique incumplir la distribución de probabilidad. Esto explica el pico que aparece en el intervalo de los 0 a 10 ms, pues se corresponde con muestras del intervalo de los 10 a 20 ms que se tuvieron que adelantar. Y la cola del margen izquierdo está formada por muestras que se retrasaron lo suficiente como para sobrepasar los 50 ms de espera.

No obstante, las muestras que no se corresponden con la distribución están en torno al 5 % del total, y no alteran demasiado la media y la varianza de los valores teóricos, como se puede comprobar a la vista de las tablas 3.4, 3.13, 3.22 y 3.31 (en algunos

casos es prácticamente igual).

Grupo 2

Se analizan los datos generados a partir de una distribución uniforme con $v_{min} = 0{,}5 \cdot 10^6$ μs y $v_{max} = 1{,}5 \cdot 10^6$ μs. Estas características se cumplen para el tiempo de espera entre peticiones de la prueba 2, con páginas web estáticas y dinámicas, y en Internet.
Se han obtenido 11 980 muestras con las que se ha representado el histograma de 60 clases que se muestra en la figura 4.2.

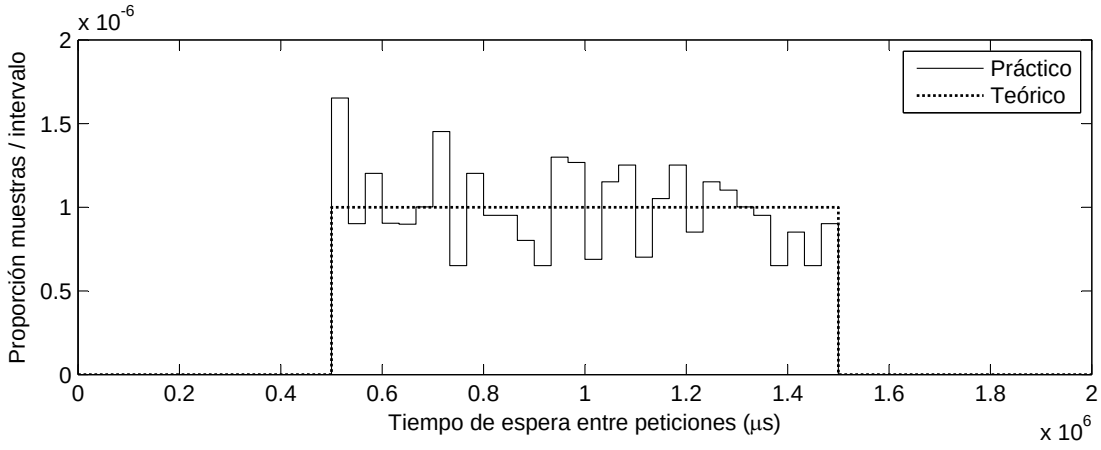

Figura 4.2: Histograma para validar el grupo de distribuciones n.º 2.

En la figura anterior se ha representado con líneas de puntos la función teórica, cuya ecuación se muestra a continuación:

$$f(x) = \begin{cases} 1 \cdot 10^{-6} & \text{para } 0{,}5 \cdot 10^6 \leq x \leq 1{,}5 \cdot 10^6, \\ 0 & \text{para cualquier otro punto} \end{cases} \tag{4.2}$$

En este caso, a diferencia del anterior, todas las muestras se encuentran dentro del intervalo previsto (entre los 0,5 y los 1,5 segundos). La distribución de las muestras parece adaptarse bien a una distribución uniforme. Hay que tener en cuenta que la media del tiempo de espera entre peticiones es 33 veces superior al caso anterior, lo que permite al procesador del cliente controlar mejor los intervalos.

Grupo 3

Se analizan los datos generados a partir de una distribución uniforme con $v_{min} = 0$ bytes y $v_{max} = 400$ bytes. Estas características se cumplen para el tamaño de los objetos de

la prueba 2, con páginas web estáticas, y en localhost, red local e Internet.
Puesto que se ha utilizado una misma semilla para generar los valores aleatorios de todas las ejecuciones, sólo se dispone de un conjunto de 50 muestras. Con ellas se ha representado el histograma de 10 clases que se muestra en la figura 4.3.

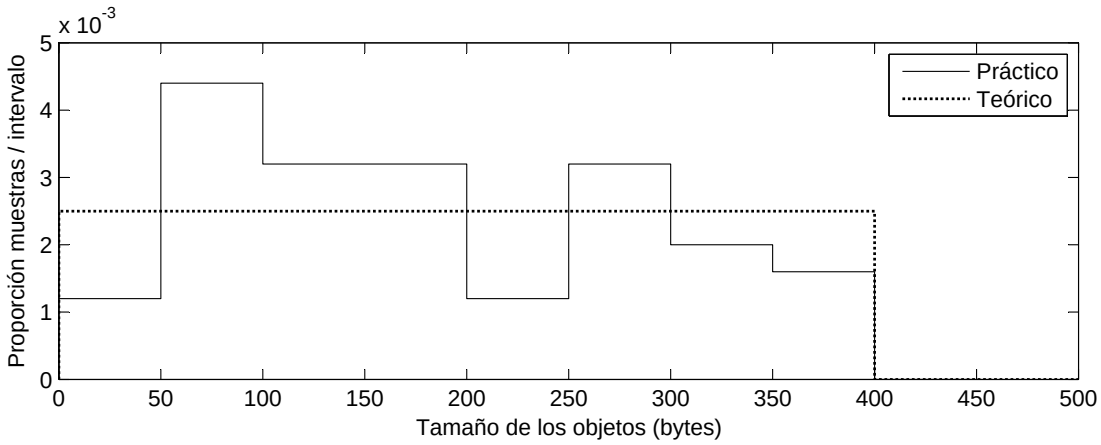

Figura 4.3: Histograma para validar el grupo de distribuciones n.º 3.

La línea de puntos de la figura anterior representa los valores teóricos de la distribución uniforme, que se obtienen con la siguiente función:

$$f(x) = \begin{cases} 2{,}5 \cdot 10^{-3} & \text{para } 0 \leq x \leq 400, \\ 0 & \text{para cualquier otro punto} \end{cases} \tag{4.3}$$

A pesar de que 50 muestras pueden no ser representativas, se aprecia cómo el histograma se adapta bien a la distribución. En las tablas 3.4, 3.22 y 3.42 se indica que el valor medio de las muestras es de 189 (cercano a los 200 que teóricamente deberían salir), con una desviación estándar de 108.

Grupo 4

Se analizan los datos generados a partir de una distribución uniforme con $v_{min} = 0$ objetos y $v_{max} = 60$ objetos. Estas características se cumplen para el número de objetos por página de la prueba 2, con páginas web estáticas, y en localhost, red local e Internet.
Como en el caso anterior, al haberse usado la misma semilla para todas las ejecuciones, se dispone de 50 muestras con las que se ha representado el histograma de 10 clases representado en la figura 4.4.

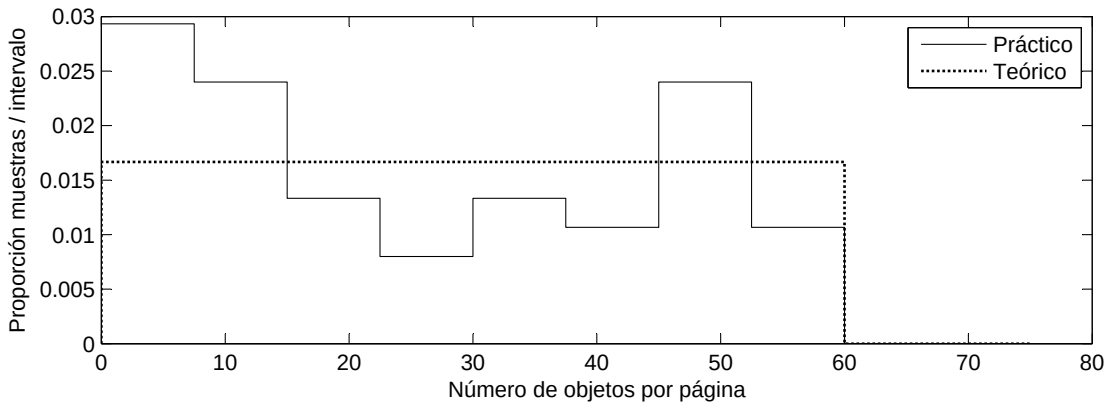

Figura 4.4: Histograma para validar el grupo de distribuciones n.º 4.

La línea de puntos representa los valores teóricos de la distribución, que se obtienen con la siguiente función:

$$f(x) = \begin{cases} 16{,}67 \cdot 10^{-3} & \text{para } 0 \leq x \leq 60, \\ 0 & \text{para cualquier otro punto} \end{cases} \tag{4.4}$$

Las 50 muestras que forman el histograma se adaptan bien a la distribución uniforme. En las tablas 3.4, 3.22 y 3.42 se indica que el número medio de objetos por página es de 26 (cercano a los 30 que teóricamente deberían salir), con una desviación estándar de 19.

4.2.2. Distribución exponencial

Grupo 5

Se analizan los datos generados a partir de una distribución exponencial con $\mu = 3 \cdot 10^4$ μs. Estas características se cumplen para el tiempo de espera entre peticiones de las pruebas 3, 4 y 7, con páginas web estáticas y dinámicas, y en localhost y red local. En total se han obtenido 2 399 880 muestras con las que se ha representado el histograma de 100 clases que se muestra en la figura 4.5.

En la figura anterior se ha superpuesto la función teórica de la distribución exponencial con una línea de puntos. La función, obtenida a partir de la ecuación 2.8, es la siguiente:

$$f(x) = \begin{cases} 3,33 \cdot 10^{-5}\, e^{-3,33 \cdot 10^{-5}\, x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{4.5}$$

Se puede comprobar cómo el histograma se adapta bien al valor teórico, aunque sólo

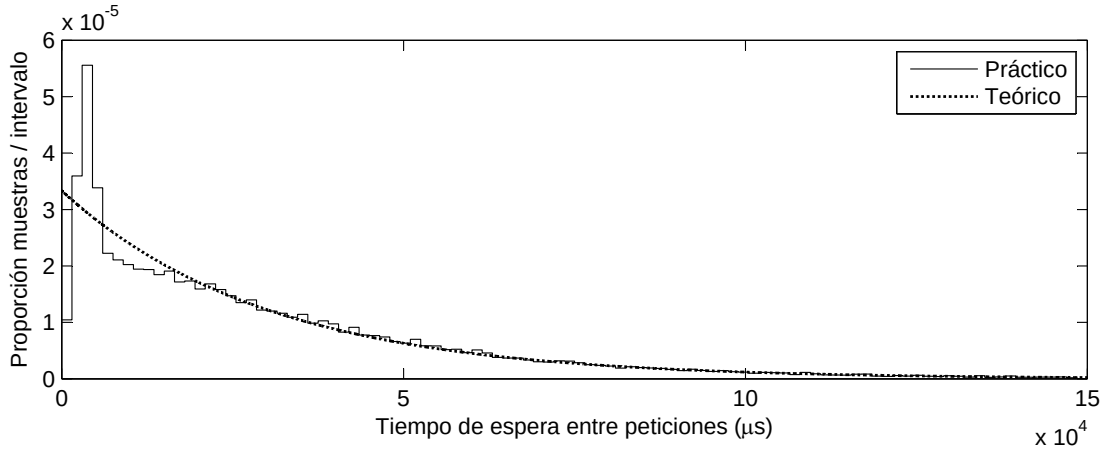

Figura 4.5: Histograma para validar el grupo de distribuciones n.º 5.

a partir de tiempos de espera superiores a los 0,02 segundos. Para tiempos de espera comprendidos entre 2 y 10 ms aparece un pico; y para valores entre cero y dos milisegundos, un mínimo relativo.

Esto ocurre porque para el procesador es muy difícil generar peticiones con un tiempo de separación entre ellas tan próximo a cero. Los retrasos añaden unos milisegundos a la espera, que al acumularse con las muestras legítimas del intervalo de los 2 a los 10 ms, crean el pico que se ve en la figura.

Las muestras que no se corresponden con la distribución no superan el 10 % del total, por lo que su media y varianza no difiere demasiado de los valores teóricos. Esto se puede comprobar, por ejemplo, en las tablas 3.5, 3.6, 3.9 o las de cualquier prueba que haya empleado una distribución exponencial con $\mu = 3 \cdot 10^4$ para definir el tiempo de espera entre peticiones.

Grupo 6

Se analizan los datos generados a partir de una distribución exponencial con $\mu = 1 \cdot 10^6$ μs. Estas características se cumplen para el tiempo entre peticiones de las pruebas 3, 4 y 7, con páginas web estáticas y dinámicas, y en Internet.

Se han obtenido 35 940 muestras con las que se ha representado el histograma de 60 clases que se muestra en la figura 4.6.

La función teórica, que también se representa en la figura anterior mediante una línea de puntos, es la siguiente:

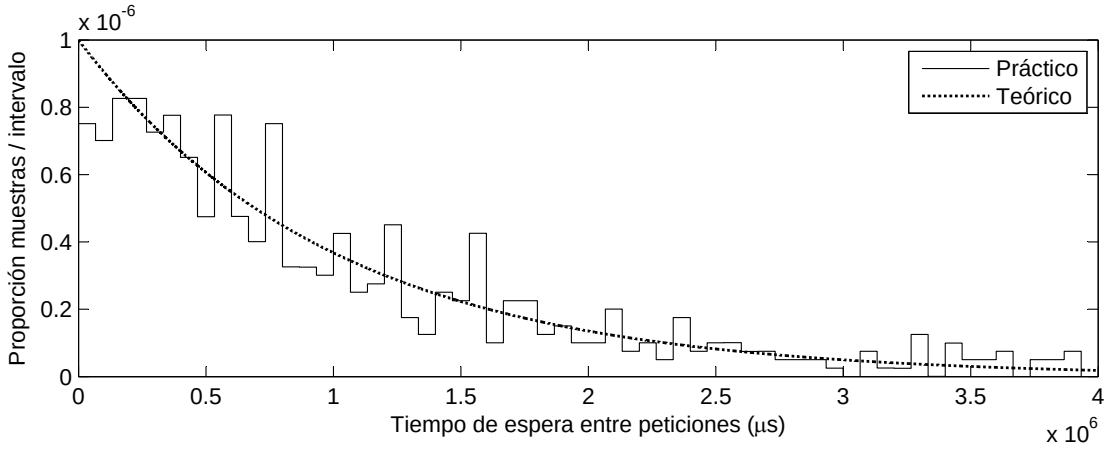

Figura 4.6: Histograma para validar el grupo de distribuciones n.º 6.

$$f(x) = \begin{cases} 1 \cdot 10^{-6}\, e^{-1 \cdot 10^{-6}\, x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{4.6}$$

En este caso, en el histograma no aparecen picos que destaquen por encima de los demás; la distribución de las muestras se adapta bien a la función teórica. El procesador del cliente puede controlar los tiempos de espera mejor que en el caso anterior porque su valor medio es 33 veces superior.

Grupo 7

Los datos analizados son los generados a partir de una distribución exponencial con $\mu = 200$ bytes. Estas características se cumplen para el tamaño de los objetos de la prueba 3, con páginas web estáticas, y en localhost, red local e Internet.
Como se ha utilizado la misma semilla para generar los valores aleatorios de todas las ejecuciones, sólo se dispone de 50 muestras diferentes. Con ellas se ha representado el histograma de 10 clases que se muestra en la figura 4.7.
La línea de puntos de la figura anterior representa los valores teóricos de la distribución uniforme, que se obtienen con la siguiente función:

$$f(x) = \begin{cases} 5 \cdot 10^{-3}\, e^{-5 \cdot 10^{-3}\, x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{4.7}$$

Aunque 50 muestras son pocas, se aprecia cómo el histograma se adapta bien a la distribución. En las tablas 3.5, 3.23 y 3.43 se indica que el valor medio de las muestras es de 193 (cercano a los 200 que teóricamente deberían salir), con una desviación

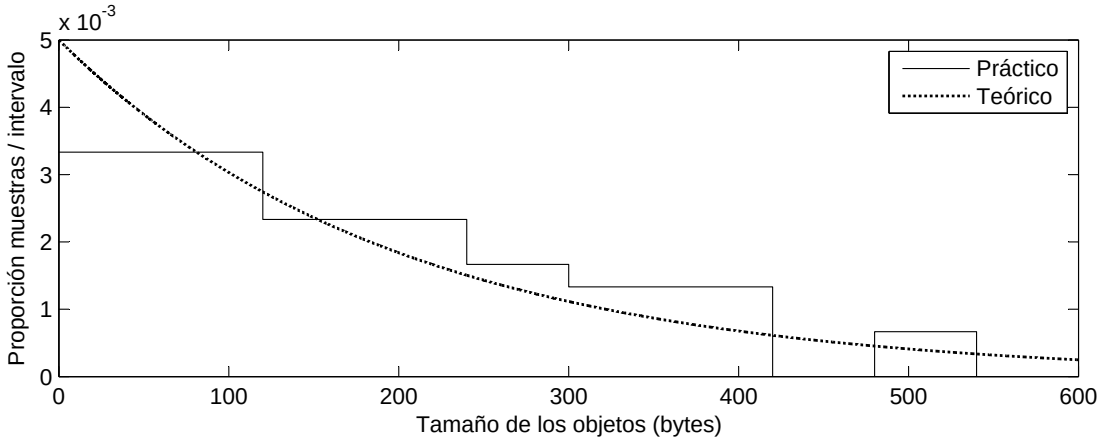

Figura 4.7: Histograma para validar el grupo de distribuciones n.º 7.

estándar de 150.

Grupo 8

Los datos analizados son los generados a partir de una distribución exponencial con $\mu = 30$ objestos. Estas características se cumplen para el número de objetos por página de la prueba 3, con páginas web estáticas, y en localhost, red local e Internet.
Como en el caso anterior, sólo se dispone de 50 muestras con las que se ha representado el histograma de 10 clases representado en la figura 4.8.

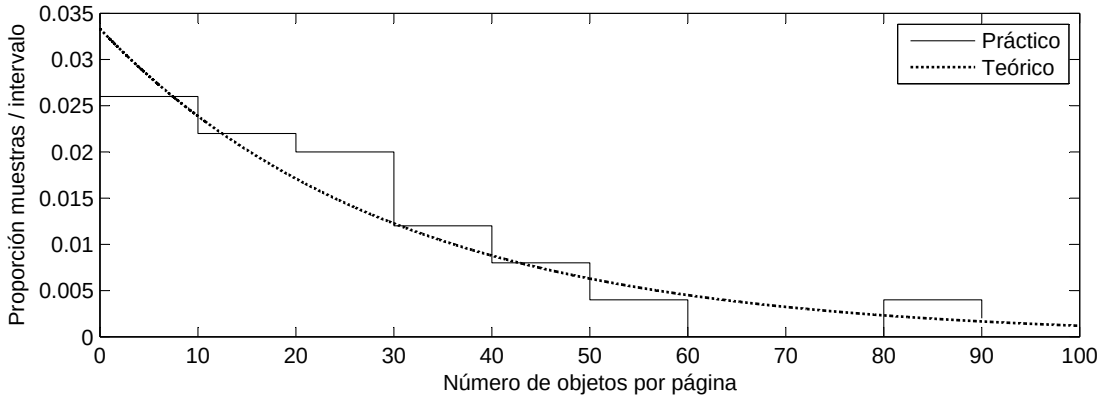

Figura 4.8: Histograma para validar el grupo de distribuciones n.º 8.

La línea de puntos representa los valores teóricos de la distribución, que se obtienen con la siguiente función:

$$f(x) = \begin{cases} 3,33 \cdot 10^{-2}\, e^{-3,33 \cdot 10^{-2}\, x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{4.8}$$

Las 50 muestras que forman el histograma se adaptan bien a la distribución exponencial. En las tablas 3.5, 3.23 y 3.43 se indica que el número medio de objetos por página es de 27 (cercano a los 30 que teóricamente deberían salir), con una desviación estándar de 27.

4.2.3. Distribución hiperexponencial

Grupo 9

Se analizan los datos generados a partir de una distribución hiperexponencial con $\mu_1 = 25 \cdot 10^3$ μs, $\mu_2 = 1 \cdot 10^5$ μs y $p_1 = 14/15$. Estas características se cumplen para el tiempo de espera entre peticiones de las pruebas 6 y 9, con páginas web estáticas y dinámicas, y en localhost y red local.
En total se han obtenido 1 599 920 muestras con las que se ha representado el histograma de 100 clases que se muestra en la figura 4.9.

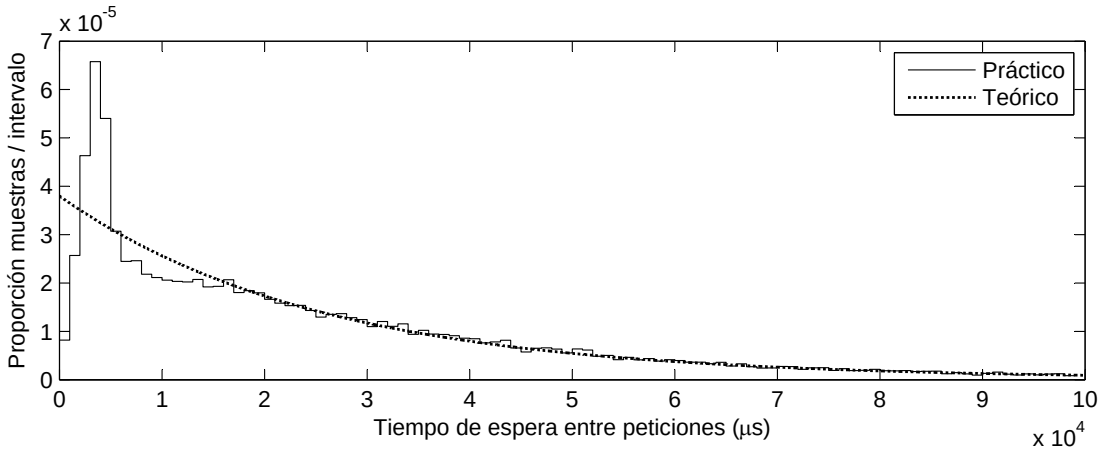

Figura 4.9: Histograma para validar el grupo de distribuciones n.º 9.

En la figura anterior se ha superpuesto la función teórica de la distribución hiperexponencial con una línea de puntos. La función, obtenida a partir de la ecuación 2.12, es la siguiente:

$$f(x) = \begin{cases} 3,7333 \cdot 10^{-5}\, e^{-4 \cdot 10^{-5}\, x} + 6,6667 \cdot 10^{-7}\, e^{-1 \cdot 10^{-5}\, x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{4.9}$$

El histograma tiene un comportamiento similar al de la distribución exponencial. Se adapta bien al valor teórico, aunque sólo a partir de tiempos de espera superiores a los 0,02 segundos. Para tiempos de espera comprendidos entre 2 y 10 ms aparece un pico; y para valores entre cero y dos milisegundos, un mínimo relativo.
Como ya se explicó, esto ocurre porque al intentar generar peticiones con un tiempo de separación cercano a cero se producen retrasos, que se acumulan en el intervalo de los 2 a los 10 ms.
Las muestras que no se corresponden con la distribución no superan el 10 % del total, por lo que su media y varianza no difiere demasiado de los valores teóricos. Esto se puede comprobar, por ejemplo, en las tablas 3.8, 3.11 o las de cualquier prueba que haya empleado una distribución hiperexponencial con las características enunciadas en este caso.

Grupo 10

Se analizan los datos generados a partir de una distribución hiperexponencial con $\mu_1 = 0{,}5 \cdot 10^6$ μs, $\mu_2 = 2 \cdot 10^6$ μs, $p_1 = 2/3$. Estas características se cumplen para el tiempo entre peticiones de las pruebas 6 y 9, con páginas web estáticas y dinámicas, y en Internet.
Se han obtenido 23 960 muestras con las que se ha representado el histograma de 60 clases que se muestra en la figura 4.10.

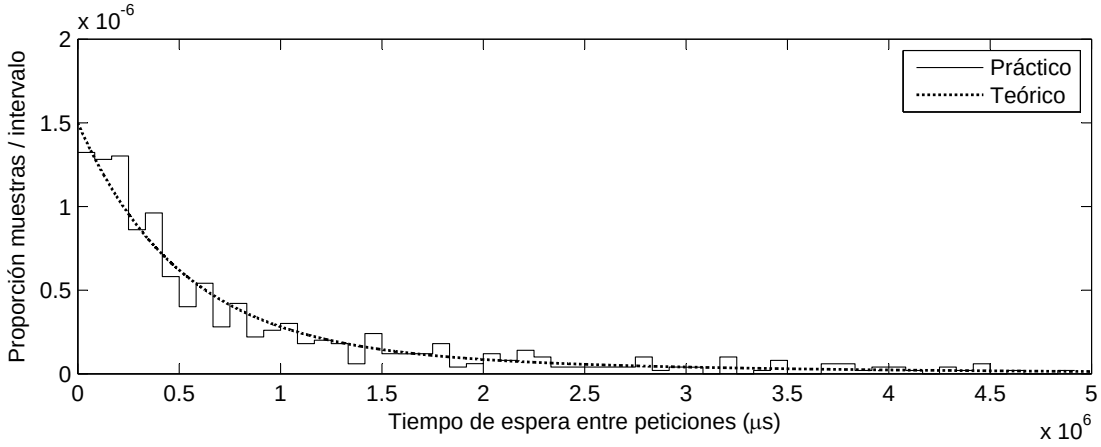

Figura 4.10: Histograma para validar el grupo de distribuciones n.º 10.

En la figura anterior se ha superpuesto la función teórica de la distribución hiperexponencial con una línea de puntos. Esta función es la siguiente:

$$f(x) = \begin{cases} 1{,}3333 \cdot 10^{-6}\, e^{-2\cdot 10^{-6}\, x} + 1{,}6667 \cdot 10^{-7}\, e^{-0{,}5\cdot 10^{-6}\, x} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases} \tag{4.10}$$

El histograma se adapta bastante bien a la función teórica. El procesador del equipo cliente puede controlar los tiempos de espera mejor que en el caso anterior porque su valor medio es 33 veces superior.

4.2.4. Distribución de Pareto

Grupo 11

Los datos que se analizan en este grupo son los generados a partir de una distribución de Pareto con $x_m = 18 \cdot 10^3$ μs y $k = 2{,}5$. Estas características se cumplen para el tiempo de espera entre peticiones de las pruebas 5 y 8, con páginas web estáticas y dinámicas, y en localhost y red local.
En total se han obtenido 1 599 920 muestras con las que se ha representado el histograma de 100 clases que se muestra en la figura 4.11.

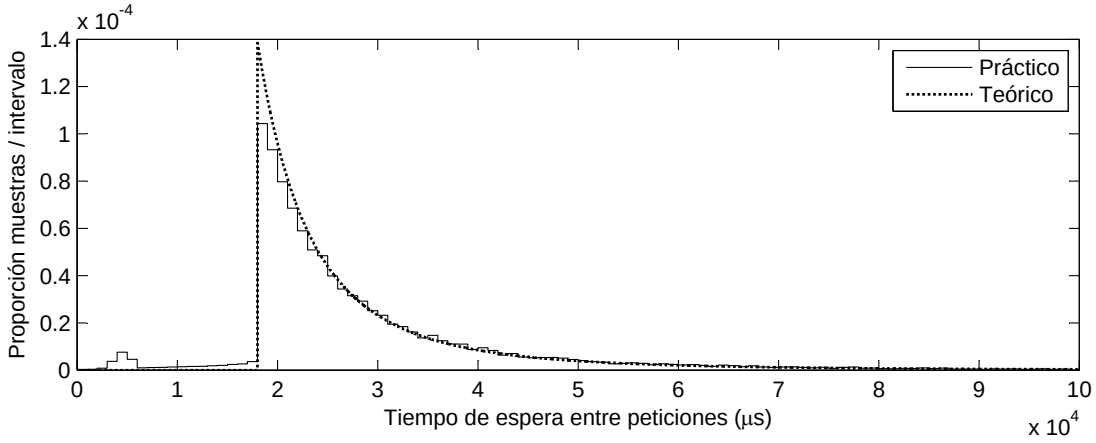

Figura 4.11: Histograma para validar el grupo de distribuciones n.º 11.

En la figura se ha superpuesto la función teórica de la distribución de Pareto con una línea de puntos. La función, obtenida a partir de la ecuación 2.16, es la siguiente:

$$f(x) = \begin{cases} \frac{1{,}0867\cdot 10^{11}}{x^{3{,}5}} & \text{para } x \geq 18 \cdot 10^3 \\ 0 & \text{para } x < 18 \cdot 10^3 \end{cases} \tag{4.11}$$

Tal y como ocurría con el primer caso de la distribución uniforme, el histograma se ajusta bastante bien al resultado teórico salvo por un aspecto: un pequeño pico en el

intervalo comprendido entre los 0 y 5 ms que no debería aparecer. Como ya se explicó en la sección 4.2.1, si el tiempo de espera de una petición se alarga más de lo debido por un retraso, el tiempo de espera de la siguiente petición se acorta para cumplir los plazos, aunque esto implique incumplir la distribución de probabilidad.
En este caso, el pico es mucho menor que en la distribución uniforme porque en aquélla el límite inferior estaba situado en los 10 ms, mientras que con la distribución de Pareto el límite inferior se sitúa en los 18 ms, casi el doble. El procesador del cliente tiene más margen de maniobra para controlar el tiempo de espera de las peticiones.
Las muestras que no se corresponden con la distribución representan menos de un 2 % del total, por lo que prácticamente no alteran la media ni la varianza de los valores teóricos, como se puede comprobar observando las tablas 3.7, 3.16, 3.25 y 3.34 (para las pruebas etiquetadas con el número 5), o las tablas 3.10, 3.19, 3.28 y 3.37 (para las pruebas etiquetadas con el número 8).

Grupo 12

Se analizan los datos generados a partir de una distribución de Pareto con $x_m = 6{\cdot}10^5$ μs y $k = 2{,}5$. Estas características se cumplen para el tiempo entre peticiones de las pruebas 5, y 8, con páginas web estáticas y dinámicas, y en Internet.
Se han obtenido 23 960 muestras con las que se ha representado el histograma de 60 clases que se muestra en la figura 4.12.

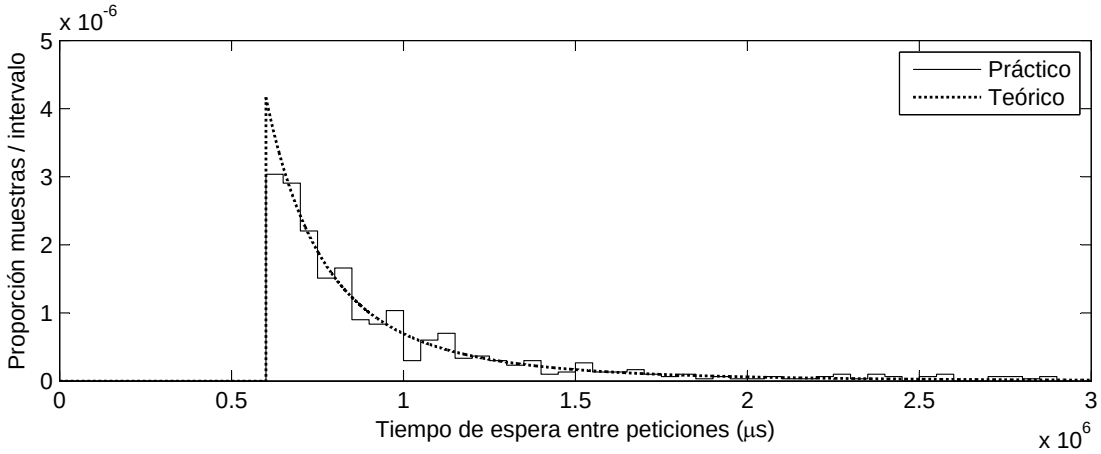

Figura 4.12: Histograma para validar el grupo de distribuciones n.º 12.

La función teórica, que también se representa en la figura mediante una línea de puntos, es la siguiente:

$$f(x) = \begin{cases} \frac{6{,}9714 \cdot 10^{14}}{x^{3,5}} & \text{para } x \geq 6 \cdot 10^5 \\ 0 & \text{para } x < 6 \cdot 10^5 \end{cases} \tag{4.12}$$

En este caso, en el histograma no aparecen picos que destaquen por encima de los demás; la distribución de las muestras se adapta bien a la función teórica. El procesador del cliente puede controlar los tiempos de espera mejor que en el caso anterior porque su valor medio es 33 veces superior.

Grupo 13

Los datos analizados son los generados a partir de una distribución de Pareto con $x_m = 57$ bytes y $k = 1{,}4$. Estas características se cumplen para el tamaño de los objetos de las pruebas 4, 5 y 6, con páginas web estáticas, y en localhost, red local e Internet.

Al haberse utilizado la misma semilla para generar los valores aleatorios de todas las ejecuciones, sólo se dispone de 50 muestras diferentes con las que se ha representado el histograma de 20 clases que se muestra en la figura 4.13.

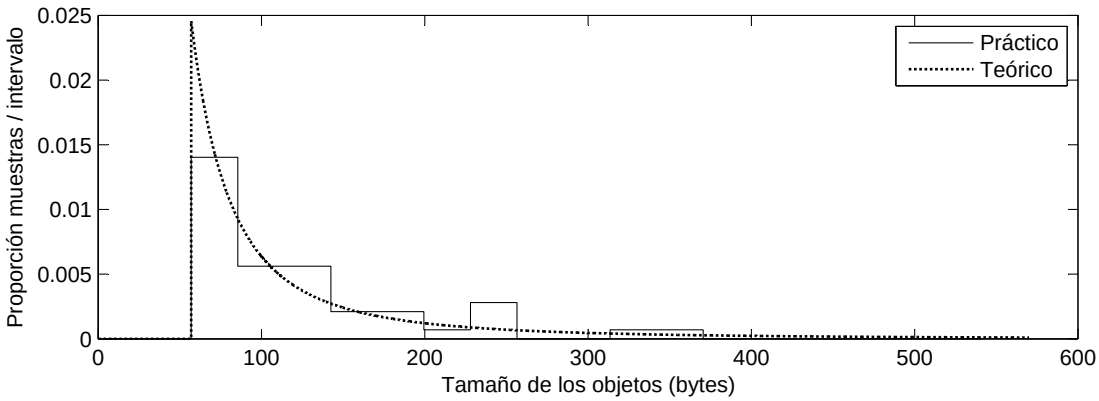

Figura 4.13: Histograma para validar el grupo de distribuciones n.º 13.

La línea de puntos de la figura representa los valores teóricos de la distribución de Pareto, que se obtienen con la siguiente función:

$$f(x) = \begin{cases} \frac{402{,}12}{x^{2,4}} & \text{para } x \geq 57 \\ 0 & \text{para } x < 57 \end{cases} \tag{4.13}$$

Aunque 50 muestras no son suficientes, se aprecia cómo el histograma se adapta bien a la distribución. En las tablas 3.6, 3.7, 3.8 y equivalentes se indica que el valor medio de las muestras es de 134, con una desviación estándar de 101.

Grupo 14

Los datos analizados son los generados a partir de una distribución de Pareto con $x_m = 18$ objetos y $k = 2{,}5$. Estas características se cumplen para el número de objetos por página de la pruebas 7, 8 y 9, con páginas web estáticas, y en localhost, red local e Internet.
Como en el caso anterior, sólo se dispone de 50 muestras con las que se ha representado el histograma de 20 clases representado en la figura 4.14.

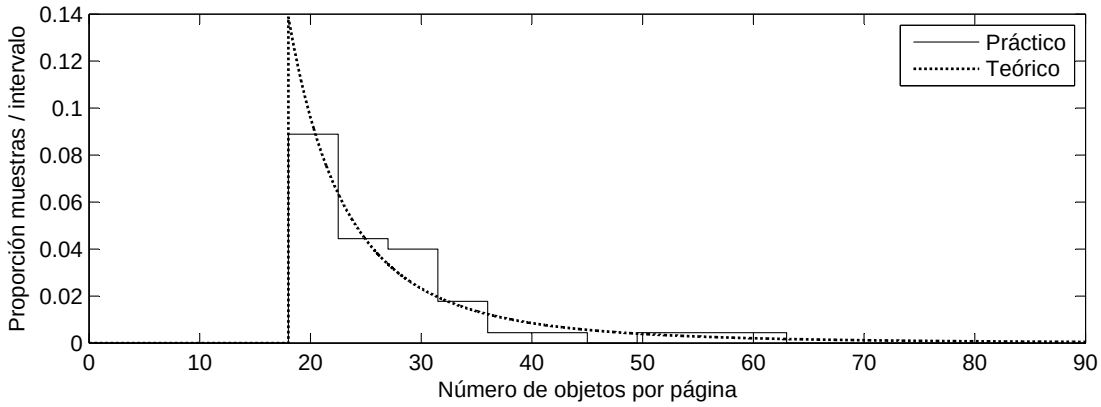

Figura 4.14: Histograma para validar el grupo de distribuciones n.º 14.

La línea de puntos representa los valores teóricos de la distribución, que se obtienen con la siguiente función:

$$f(x) = \begin{cases} \frac{3436{,}5}{x^{3,5}} & \text{para } x \geq 18 \\ 0 & \text{para } x < 18 \end{cases} \tag{4.14}$$

Las 50 muestras que forman el histograma se adaptan bien a la distribución de Pareto. En las tablas 3.9, 3.10, 3.11 o equivalentes se indica que el número medio de objetos por página es de 30, con una desviación estándar de 19.

4.2.5. Distribución lognormal

Grupo 15

Los datos analizados son los generados a partir de una distribución lognormal con $\mu = 200$ bytes y $\sigma^2 = 40\,000$ bytes^2. Estas características se cumplen para el tamaño de los objetos de las pruebas 7, 8 y 9, con páginas web estáticas, y en localhost, red local e Internet.

Se dispone de 50 muestras diferentes con las que se ha representado el histograma de 15 clases que se muestra en la figura 4.15.

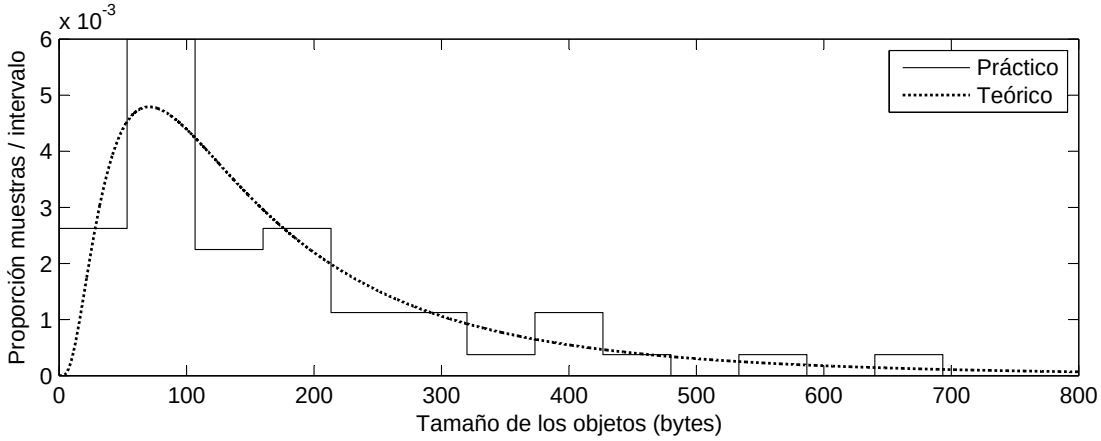

Figura 4.15: Histograma para validar el grupo de distribuciones n.º 15.

En la figura anterior se ha superpuesto la función teórica de la distribución lognormal con una línea de puntos. La función, obtenida a partir de la ecuación 2.20, es la siguiente:

$$f(x) = \begin{cases} \frac{1}{2{,}0869 \cdot x} \cdot \exp\left[-\frac{(\log x - 4{,}9517)^2}{1{,}3863}\right] & \text{para } x > 0 \\ 0 & \text{para } x \leq 0 \end{cases} \tag{4.15}$$

El histograma parece adaptarse bien a la distribución. En las tablas 3.9, 3.10, 3.11 y equivalentes se indica que el valor medio de las muestras es de 192, con una desviación estándar de ±211.

4.2.6. Valor constante

Grupo 16

Los datos que se analizan en este caso son los generados a partir de los valores constantes con $c = 3 \cdot 10^4$ μs. Estas características se cumplen para el tiempo de espera entre peticiones de la prueba 1, con páginas web estáticas y dinámicas, y en localhost y red local.

En total se han obtenido 799 960 muestras con las que se ha representado el histograma de 99 clases que representado en la figura 4.16.

En la figura destacan tres picos: uno centrado en los 0,03 segundos, que es el único que debería aparecer; y otros dos, a ambos lados del primero, separados por unos 15 microsegundos.

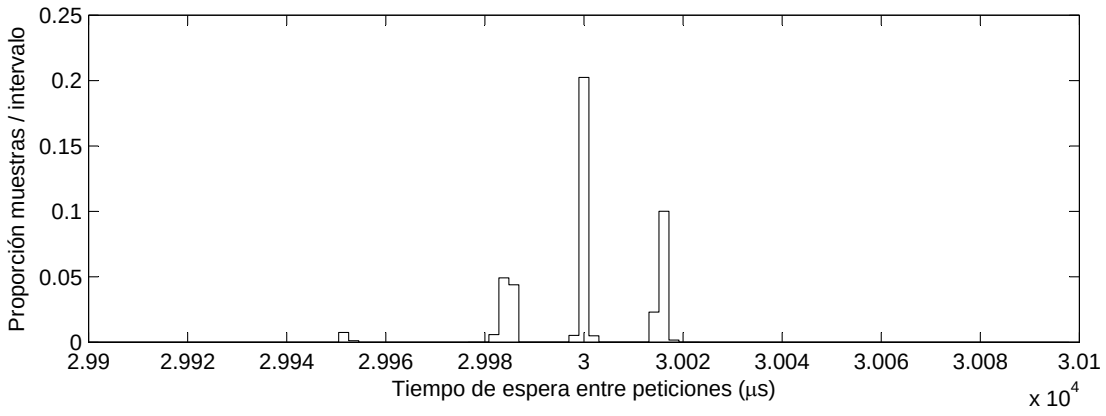

Figura 4.16: Histograma para validar el grupo de distribuciones n.º 16.

La interpretación del histograma es clara: cuando una muestra sufre un retraso (que en las ejecuciones analizadas suele estar en torno a los 15 μs) la muestra siguiente se adelanta otros 15 μs para no retrasar al resto de peticiones.
A pesar de los dos picos extra, en las tablas 3.3, 3.12, 3.21 y 3.30 se puede comprobar que el tiempo medio entre peticiones y sus variancias son muy similares a los que se tendrían que haber obtenido en teoría.

Grupo 17

Los datos analizados en este caso son los generados a partir de los valores constantes con $c = 1 \cdot 10^6$ μs. Estas características se cumplen para el tiempo de espera entre peticiones de la prueba 1, con páginas web estáticas y dinámicas, y en Internet.
En total se han obtenido 11 980 muestras con las que se ha representado el histograma de 59 clases representado en la figura 4.17.
La figura muestra un único pico centrado en 1 segundo. Las tablas 3.41 y 3.50 permiten comprobar que las peticiones se han realizado con una media y varianza muy similares a las teóricas.

4.3. Conclusiones

En general, las distribuciones de probabilidad simuladas por el benchmark se adaptan bastante bien a los valores teóricos. Los únicos casos en los que se observan muestras con un comportamiento incorrecto corresponden a pruebas con mucha carga y tiempos de espera entre peticiones muy cercanos a cero.

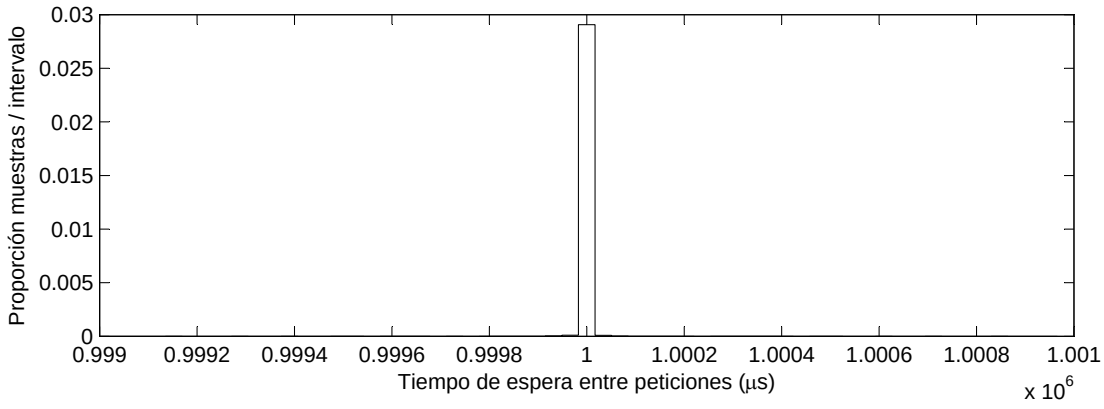

Figura 4.17: Histograma para validar el grupo de distribuciones n.º 17.

Como se ha explicado anteriormente, las muestras incorrectas son producto de pequeños retrasos del procesador, y suelen aparecer cuando éste atiende a un gran número de procesos simultáneamente (cabe recordar que el cliente controla cada petición que se realiza al servidor con un proceso diferente).
Con todo, la frecuencia de aparición de estas muestras es muy baja, y no alteran de forma drástica las características generales de las distribuciones de probabilidad de las que forman parte.

Apéndice A

CONTENIDO DEL DVD

En la parte interior de la contracubierta de esta memoria se adjunta un DVD. Su contenido se distribuye en las cinco carpetas que se enumeran a continuación:

1. `Benchmark`. Esta carpeta contiene los siete ficheros necesarios para instalar el benchmark en un equipo:

   1.1. `gendb2.txt`. Proporciona la base de datos necesaria para generar páginas web dinámicas.

   1.2. `makefile`. Permite la compilación de los programas estableciendo las dependencias necesarias.

   1.3. `qb.c`. Código fuente del programa generador de carga (véase la sección 2.6).

   1.4. `qgen.c`. Código fuente del programa generador de páginas web (véase la sección 2.7).

   1.5. `qlib.c`. Forma parte del módulo `qlib`. En este fichero se desarrollan las funciones declaradas en qlib.h (véase la sección 2.5.2).

   1.6. `qlib.h`. Forma parte del módulo `qlib`. Es el fichero de cabecera donde se encuentran las constantes, tipos y prototipos de las funciones visibles fuera del módulo (véase la sección 2.5.1).

   1.7. `qmon.c`. Código fuente del programa monitor de procesos (véase la sección 2.8).

2. `C2 - Implementación`. Esta carpeta contiene cinco ficheros en código M que implementan los algoritmos para generar muestras con las distribuciones de probabilidad descritas en el capítulo 2:

2.1. `distribE.m`. Distribución exponencial (véase la sección 2.3.4).

2.2. `distribH.m`. Distribución hiperexponencial (véase la sección 2.3.5).

2.3. `distribL.m`. Distribución lognormal (véase la sección 2.3.7).

2.4. `distribP.m`. Distribución de Pareto (véase la sección 2.3.6).

2.5. `distribU.m`. Distribución uniforme (véase la sección 2.3.3).

3. `C3 - Resultados`. Esta carpeta contiene los programas usados para obtener las gráficas y estadísticas que se muestran en el capítulo 3, y los ficheros resultantes de las diferentes ejecuciones:

3.1. `Datos - Internet - Dinámicas.rar`. Archivo comprimido con los 360 ficheros resultantes de la ejecución del benchmark con páginas dinámicas en Internet (véase la sección 3.5.2).

3.2. `Datos - Internet - Estáticas.rar`. Archivo comprimido con los 270 ficheros resultantes de la ejecución del benchmark con páginas estáticas en Internet (véase la sección 3.5.1).

3.3. `Datos - Localhost - Dinámicas.rar`. Archivo comprimido con los 360 ficheros resultantes de la ejecución del benchmark con páginas dinámicas en localhost (véase la sección 3.3.2).

3.4. `Datos - Localhost - Estáticas.rar`. Archivo comprimido con los 270 ficheros resultantes de la ejecución del benchmark con páginas estáticas en localhost (véase la sección 3.3.1).

3.5. `Datos - Red local - Dinámicas.rar`. Archivo comprimido con los 360 ficheros resultantes de la ejecución del benchmark con páginas dinámicas en red local (véase la sección 3.4.2).

3.6. `Datos - Red local - Estáticas.rar`. Archivo comprimido con los 270 ficheros resultantes de la ejecución del benchmark con páginas estáticas en red local (véase la sección 3.4.1).

3.7. `qbucle2.pl`. *Script* en Perl para automatizar las diez ejecuciones de cada una de las nueve pruebas realizadas por el benchmark.

3.8. `qestad.m`. Calcula estadísticas de varios parámetros a partir de los ficheros `csv` de una prueba, y muestra los resultados en el formato de tabla de LaTeX (véase la sección 3.2.2).

3.9. `qgraf2.m`. Representa el 95 percentil de los tiempos de respuesta de las páginas web solicitadas en cada prueba y el porcentaje del consumo medio del procesador del servidor (véase la sección 3.2.1).

3.10. `Resultados Matlab.mat`. Contiene las variables de Matlab que permiten reproducir las gráficas del capítulo 3.

4. `C4 - Validación`. Esta carpeta contiene los programas y las variables de Matlab usadas para obtener las gráficas que se muestran en el capítulo 4:

4.1. `qval.m`. Representa los valores de un vector en un histograma de área igual a 1 y superpone la curva de una distribución de probabilidad seleccionada por el usuario.

4.2. `resta.m`. Resta cada elemento de un vector con su elemento inmediatamente anterior.

4.3. `tomaN.m`. Lee de un fichero `csv` la variable `tiempoInicioPag` y la convierte en un vector de Matlab.

4.4. `tomaNbig.m`. Lee de los diez ficheros `csv` generados por una misma prueba la variable `tiempoInicioPag` y la convierte en un vector de Matlab.

4.5. `tomaO.m`. Lee de un fichero `csv` la variable `tamanoObjetosPagina` y la convierte en un vector de Matlab.

4.6. `tomaP.m`. Lee de un fichero `csv` la variable `objetosPagina` y la convierte en un vector de Matlab.

4.7. `Validación Matlab.mat`. Contiene las variables de Matlab que permiten reproducir las gráficas del capítulo 4.

5. `Memoria`. Esta carpeta contiene dos ficheros:

5.1. `Memoria.pdf`. El fichero `pdf` de esta memoria.

5.2. `Memoria (hiperenlaces).pdf`. El fichero `pdf` de esta memoria con hiperenlaces.

Apéndice B

INSTALACIÓN

En este apéndice se detallan los pasos para instalar correctamente el benchmark en una distribución de Linux (en este caso se detalla para Ubuntu 7.10, aunque para otras distribuciones el procedimiento es similar). El procedimiento descrito permite que una misma máquina pueda actuar como cliente y servidor simultáneamente. Para ejecutar el benchmark entre dos ordenadores es necesario realizar la misma instalación en cada uno de ellos (cuál actuará como cliente y cuál como servidor será una decisión que tendrá que tomar el usuario).

B.1. Instalación de paquetes

A continuación se detalla una lista con los nueve paquetes que son necesarios para ejecutar el benchmark y que no se encuentran instalados por defecto en Ubuntu. Para instalarlos en esta distribución hay que ejecutar el *Gestor de paquetes Synaptic*, que se encuentra en el menú *Sistema*, *Administración*. Para distribuciones de Linux que usen entornos de escritorio KDE, la aplicación equivalente es el *Administrador Adept*; también se puede usar la instrucción `apt-get` desde la línea de comandos. Los paquetes son:

- `build-essential`. Contiene una lista informativa de paquetes que se consideran esenciales para compilar paquetes Debian. Es necesario para ejecutar el fichero `makefile`. El compilador de C de GNU, `gcc`, ya se viene instalado por defecto en Ubuntu.
- `openssh-server`. Proporciona un servidor SSH. El cliente, `openssh-client`, ya viene instalado por defecto en Ubuntu.

- `apache2`, `libapache2-mod-php5` y `libapache2-mod-auth-mysql`. Servidor web de Apache. Se añaden dos módulos: uno para PHP5 y otro que permite la autenticación HTTP contra la información almacenada en una base de datos de MySQL.
- `mysql-server` y `mysql-client`. Servidor y cliente de MySQL.
- `php5` y `php5-mysql`. Intérprete de PHP. Se añade un módulo que permite conexiones con la base de datos MySQL desde los *scripts* PHP.

Estos paquetes y sus dependencias requieren cerca de 150 MB de espacio en disco. Durante el proceso de instalación es probable que se solicite el disco de Ubuntu 7.10, por lo que conviene tenerlo a mano. También es probable que una ventana emergente pregunte por la contraseña de `root` de MySQL: hay que dejarla en blanco y continuar con la instalación.
Una vez instalados todos los paquetes es necesario reiniciar el equipo (se ha comprobado que PHP no atiende correctamente a la base de datos hasta que no se reinicia). En caso de instalar el benchmark en una distribución de Linux diferente de Ubuntu, podría ser necesario instalar otros paquetes adicionales.

B.1.1. Comprobación de la instalación de Apache

Para comprobar que Apache se ha instalado correctamente, basta con abrir un navegador web (*Mozilla Firefox*, *Konqueror*, etc.) y escribir la dirección de *localhost*, `127.0.0.1`, en la barra de direcciones. Deberá aparecer una carpeta llamada `apache2-default`. Al hacer clic sobre ella, se mostrará el mensaje «*It works!*».

B.1.2. Comprobación de la instalación de PHP

Con PHP el procedimiento es un poco más laborioso. Se debe acceder como superusuario (mediante el comando `sudo` desde la línea de comandos) a un editor de texto (*gedit*, *Kate*, *nano*, etc.) y se debe escribir el siguiente código:

```
<? phpinfo(); ?>
```

Se debe guardar con el nombre de `prueba.php` en la carpeta protegida `/var/www`. A continuación, se abre en un navegador y se vuelve a escribir la dirección de *localhost*. Se podrá ver el fichero recién creado y, si al hacer clic sobre él se muestra una tabla con información sobre el sistema, entonces la instalación ha sido correcta.

B.2. Configuración de la base de datos

Para que se puedan servir páginas dinámicas es necesario instalar la base de datos proporcionada en el archivo `gendb2.txt`. Para ello hay que seguir una serie de pasos. Primero, hay que crear un usuario con el que se pueda acceder a MySQL:

1. Se entra en MySQL en modo administrador: `sudo mysql`
2. Se crea un nuevo usuario mediante la instrucción:

   `GRANT USAGE ON *.* TO <usuario> IDENTIFIED BY '<contraseña>';`

   donde `<usuario>` es el nombre de usuario que se desea tener y `<contraseña>` la contraseña. Es importante introducir el punto y coma al final de la instrucción. Se recomienda que el nombre de usuario sea `root`. Esto reducirá el número de parámetros que se tendrán que introducir al benchmark desde la línea de comandos. También se recomienda dejar la contraseña en blanco (es decir, escribiendo `IDENTIFIED BY ''`), o de lo contrario habrá que escribirla de forma visible desde la consola.
3. Se dan todos los privilegios al nuevo usuario creado:

   `GRANT ALL ON *.* TO <usuario>;`
4. Se sale mediante la instrucción `exit`.

A continuación hay que acceder a MySQL. Para ello se debe ejecutar el comando:
`mysql -h localhost -u` *`usuario`* `-p`
El programa pedirá la contraseña anteriormente proporcionada; si se ha dejado en blanco, bastará con pulsar *Enter*.
Por último, se deben introducir las siguientes instrucciones (sin olvidar los puntos y coma finales). Se recomienda que el nombre de la base de datos sea `qgen` y el de la tabla `qgendb`, ya que son los nombres por defecto que utiliza el benchmark. Cualesquiera otros nombres para la base de datos y la tabla deberán ser introducidos como parámetros por la línea de comandos cada vez que se ejecute el benchmark:

1. `DROP DATABASE IF EXISTS qgen;`
2. `CREATE DATABASE qgen;`
3. `USE qgen;`

4. `CREATE TABLE qgendb (id varchar(1), c1024 varchar(1024),`
   `c512 varchar(512), c256 varchar(256), c128 varchar(128),`
   `c64 varchar(64), c32 varchar(32), c16 varchar(16), c8 varchar(8),`
   `c4 varchar(4), c2 varchar(2), c1 varchar(1));`

5. `LOAD DATA LOCAL INFILE '/<ruta>/gendb2.txt'`
   `INTO TABLE qgen.qgendb;`
   donde `<ruta>` es la dirección completa del fichero `gendb2.txt`.

Para comprobar que se ha hecho bien, la ejecución de este comando:
`SELECT c16 FROM qgendb WHERE id = "2";`
tiene que dar el siguiente resultado:

```
+------------------+
| c16              |
+------------------+
| us ac velit orna |
+------------------+
1 row in set (0.00 sec)
```

El comando hace que se muestre por pantalla el dato determinado por el segundo registro y el campo identificado como `c16`.
Para salir del entorno de MySQL sólo hay que escribir `exit`.

B.3. Compilación de los ficheros fuente

Para instalar los tres programas que componen el benchmark, es conveniente copiar la carpeta "`Benchmark`" del DVD al ordenador. Esta carpeta contiene los ficheros fuente y un fichero `makefile`. Los pasos que se deben seguir son:

1. Desde la línea de comandos, cambiar el directorio de trabajo por el directorio que contiene los ficheros fuente.

2. Escribir la orden `make -B`, que ejecutará las instrucciones contenidas en el fichero `makefile`. Durante la ejecución, se solicitará la contraseña de administrador para guardar en la carpeta `/bin` los ficheros ejecutables. Es necesario introducirla para que los programas que componen el benchmark puedan estar disponibles desde cualquier directorio de trabajo.

A partir de este momento el benchmark ya es funcional. Se puede comprobar porque al teclear `qb` desde consola, en cualquier directorio de trabajo, debe aparecer el mensaje de ayuda del benchmark.

B.4. Configuración de SSH

Cada vez que se realiza una conexión SSH con el usuario de una máquina remota es necesario escribir su contraseña. Por cada ejecución del benchmark se realizan tres conexiones SSH con el usuario que controla el servidor web en la máquina remota. Si se pretende que el benchmark funcione de una manera lo más automática posible, escribir tres veces la misma contraseña resta bastante eficiencia al proceso.
Existe una forma de obtener acceso al servidor mediante SSH sin tener que teclear la contraseña del usuario remoto, y es la autenticación mediante claves RSA. Las instrucciones para conseguirla son sencillas, pero antes de llevarlas a cabo hay que tener claro qué ordenadores van a hacer de cliente y de servidor cuando se ejecute el benchmark:

1. Acceder como `root` a la línea de comandos: `su root`

2. Generar un par de claves en la máquina cliente: la pública y la privada. Para hacerlo se escribe:

   ```
   ssh-keygen -t rsa
   ```

   Primero pregunta en qué ruta se prefiere guardar la clave. Sólo se presionará *Enter* para aceptar la que ofrece por defecto. Luego pregunta por una frase secreta, y que se debe dejar en blanco, ya que de lo contrario pedirá esa frase cuando se intente conectar.

3. Copiar la clave pública al servidor en la carpeta del usuario con el cual se quiere acceder sin que pida contraseña. Como se ha comentado antes, lo mejor es acceder como `root`. Para mandar la copia al *home* de `root` del servidor desde el cliente:

   ```
   scp /root/.ssh/id_rsa.pub root@host:/root/.ssh/authorized_keys2
   ```

Ahora, cada vez que el usuario `root` del equipo cliente quiera acceder mediante SSH al equipo servidor como usuario `root`, no se le requerirá la contraseña. El procedimiento se debe repetir por cada combinación de usuarios y equipos diferente que se quiera permitir.

Apéndice C

MANUAL DE USO

Los programas cuya guía de uso se detalla en las siguientes secciones siguen las pautas de sintaxis de aplicaciones que marca *The Single UNIX Specification* [14]. Estas pautas establecen el nombre que deben tener las aplicaciones y las especificaciones que deben seguir las opciones, los argumentos y los operandos que se les puedan pasar a las aplicaciones desde la línea de comandos.

C.1. Sintaxis del generador de carga

```
qb -d [ -a dirección-IP ] [ -b nombre-BD ] [ -f directorio-salida ]
[ -h puerto-HTTP ] [ -m contraseña-MySQL ] -n atributos-peticiones
-o atributos-objetos -p atributos-páginas [ -r fichero-salida ]
[ -S puerto-SSH ] [ -s semilla ] [ -t tabla-BD ] [ -U usuario-BD ]
[ -u usuario ] [ -x ]

qb -e [ -a dirección-IP ] [ -f directorio-salida ] [ -h puerto-HTTP ]
-n atributos-peticiones -o atributos-objetos -p atributos-páginas
[ -r fichero-salida ] [ -S puerto-SSH ] [ -s semilla ] [ -u usuario ]
[ -x ]
```

- `-a dirección-IP`. Dirección IP del servidor. La dirección se debe especificar mediante la notación por puntos, y debe seguir una de las cuatro formas siguientes:

  `a.b.c.d`

```
a.b.c
a.b
a
```

Cuando se especifican las cuatro partes, cada una se interpreta como un byte de datos y se asigna, de izquierda a derecha, a los cuatro bytes de una dirección de Internet.

Cuando se especifican tres partes, la última se interpreta como una cantidad de 16 bits y se asigna a los dos bytes más a la derecha de la dirección de red. Este formato en tres partes puede ser conveniente para especificar las direcciones de red de clase B, como `128.red.host`.

Cuando se proporciona una dirección en dos partes, la última parte se interpreta como una cantidad de 24 bits y se asigna a los tres bytes más a la derecha de la dirección de red. Este formato es conveniente para especificar direcciones de red de clase A, como `red.host`.

Cuando sólo se proporciona una parte, el valor se almacena directamente en la dirección de red sin ninguna reordenación de bytes.

Todos los números proporcionados en la notación por puntos deben ser decimales, octales o hexadecimales, tal y como se especifican en el lenguaje C (es decir, un `0x` o `0X` en la parte delantera implica un número hexadecimal; un `0` en la parte delantera implica números octales; en caso contrario, el número se interpreta como decimal).

- `-b nombre-BD`. Nombre de la base de datos de trabajo de MySQL. Es necesario llamar a esta opción cuando el nombre sea distinto de `qgen`, como se explicó en la sección B.2.
- `-d`. Indica que el benchmark tiene que trabajar con páginas web dinámicas.
- `-e`. Indica que el benchmark tiene que trabajar con páginas web estáticas.
- `-f directorio-salida`. Directorio del servidor en el que el programa generador de páginas web creará los ficheros. Es necesario llamar a esta opción cuando la ruta sea distinta de `/var/www`.
- `-h puerto-HTTP`. Número de puerto por el que se realizan las peticiones HTTP. Es necesario llamar a esta opción cuando se ha configurado un puerto distinto del 80. El servidor de páginas web de Apache escucha por defecto este puerto.

- `-m contraseña-MySQL`. Contraseña con la que el usuario se conecta a la base de datos de MySQL. Es necesario llamar a esta opción si se ha definido una contraseña durante el proceso de configuración de la base de datos (véase la sección B.2).

- `-n atributos-peticiones`. Atributos de la distribución de probabilidad con la que se obtendrán los tiempos de espera entre peticiones. El modo en el que se tienen que proporcionar estos atributos se explica más adelante.

- `-o atributos-objetos`. Atributos de la distribución de probabilidad con la que se obtendrán los tamaños de los objetos que componen las páginas web. El modo en el que se tienen que proporcionar estos atributos se explica más adelante.

- `-p atributos-páginas`. Atributos de la distribución de probabilidad con la que se obtendrá el número de objetos que contiene cada página web. El modo en el que se tienen que proporcionar estos atributos se explica más adelante.

- `-r fichero-salida`. Nombre de los ficheros de salida `*.txt` y `*.cvs` obtenidos tras una ejecución correcta del programa. Si no se indica nada, el valor por defecto es `result`. Es interesante usar esta opción cuando se quieren conservar los informes de diferentes ejecuciones del programa.

- `-S puerto-SSH`. Número del puerto por el que se realizan las conexiones SSH. Es necesario llamar a esta opción cuando se ha configurado un puerto distinto del 22. OpenSSH escucha por defecto este puerto.

- `-s semilla`. Valor de la semilla del generador de números pseudoaleatorios. Debe ser un número natural comprendido entre 0 y 2147483647. Es interesante usar esta opción cuando se quieren obtener los mismos valores en diferentes ejecuciones del programa. Si no se incluye esta opción, los valores generados son siempre distintos.

- `-t tabla-BD`. Nombre de la tabla usada por la base de datos con la que trabaja MySQL. Es necesario llamar a esta opción cuando el nombre sea distinto de `qgendb`, como se explicó en la sección B.2.

- `-U usuario-BD`. Nombre de usuario de acceso a MySQL. Es necesario llamar a esta opción cuando el nombre sea distinto de `root`, como se explicó en la sección B.2.

- `-u usuario`. Nombre de usuario del equipo servidor. Es necesario llamar a esta opción cuando el nombre sea distinto de `root`.
- `-x`. Si se incluye esta opción, no se activará el programa monitor de procesos en el servidor.

Los atributos que conforman los argumentos de las opciones `-n`, `-o` y `-p` se deben escribir en un sólo bloque, sin espacios y separados por comas. El número de atributos puede variar entre 2 y 5, y se deben especificar con el siguiente orden:
El primer atributo debe ser el número de elementos que se quieren crear. En el caso de la opción `-n`, el primer atributo indica el número de peticiones que se quieren realizar al servidor; en el caso de la opción `-o`, indica el número de ficheros-objeto que se quieren crear (sólo para páginas estáticas); y en el caso de la opción `-p`, indica el número de páginas web que se quieren crear. Cuando las páginas web son dinámicas no hay que especificar este atributo con la opción `-o`.
El segundo atributo es un carácter que se escogerá dependiendo de la distribución de probabilidad que se quiera asignar. Los atributos tercero y siguientes son valores numéricos que dependen de la distribución seleccionada. En la tabla que se muestra a continuación se especifican los atributos:

Distribución	Atrib. 2	Atrib. 3	Atrib. 4	Atrib. 5
distr. exponencial	`e`	valor medio	—	—
distr. hiperexponencial	`h`	valor medio 1	valor medio 2	probab. 1
distr. lognormal	`l`	valor medio	varianza	—
distr. de Pareto	`p`	localización	forma	—
distr. uniforme	`u`	valor mínimo	valor máximo	—
valor constante	`c`	valor	—	—

Cuadro C.1: Atributos de las opciones `-n`, `-o` y `-p`.

C.2. Sintaxis del generador de páginas web

```
qgen -d [ -a dirección-IP ] [ -b nombre-BD ] [ -f directorio-salida ]
[ -m contraseña-MySQL ] -o atributos-objetos -p atributos-páginas
[ -s semilla ] [ -t tabla-BD ] [ -U usuario-BD ] [ -u usuario ]

qgen -e [ -f directorio-salida ] -o atributos-objetos
-p atributos-páginas [ -s semilla ]
```

Todas las opciones que utiliza este programa ya han sido comentadas en la sección C.1.

C.3. Sintaxis del monitor de procesos

```
qmon -d [ -r fichero-salida ]
qmon -e [ -r fichero-salida ]
```

Todas las opciones que utiliza este programa ya han sido comentadas en la sección C.1.

C.4. Recomendaciones

C.4.1. Creación del usuario root en Ubuntu

Se recomienda ejecutar el benchmark como superusuario para evitar tener que escribir contraseñas cada vez que se ejecuta el programa.
En Linux existe un cuenta de usuario especial llamada `root` (o superusuario) que actúa como la cuenta del administrador del sistema. El usuario `root` puede realizar muchas acciones que un usuario común no puede. Por defecto, la contraseña de la cuenta de `root` está bloqueada en Ubuntu. Para activarla sólo hay que escribir la siguiente orden desde la línea de comandos: `sudo -u root passwd`
A continuación se solicitará una contraseña nueva para este usuario y su confirmación.

C.4.2. Ampliación del número de descriptores de fichero

Cuando el benchmark tiene que realizar un número elevado de peticiones (más de mil) es necesario ampliar el límite máximo de descriptores de fichero que la consola puede atender por defecto.
Para conseguirlo basta con escribir la siguiente orden desde la línea de comandos antes de ejecutar el benchmark: `ulimit -n X`, donde `X` se debe sustituir por un número covenientemente grande.

Bibliografía

[1] ipoque Internet Studies: Internet Study 2007 (extended abstract).
`http://www.ipoque.com/userfiles/file/Internet_study_2007_abstract_en.pdf`

[2] Ramón Puigjaner, Juan José Serrano y Alicia Rubio. *Evaluación y explotación de sistemas informáticos*. Síntesis, 1995. ISBN 8477383162.

[3] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurements, Simulation, and Modeling*. Wiley, 1991, ISBN 0471503363.

[4] Apache HTTP server benchmarking tool.
`http://httpd.apache.org/docs/2.2/programs/ab.html`

[5] httperf homepage.
`http://www.hpl.hp.com/research/linux/httperf/`

[6] The Grinder, a Java Load Testing Framework.
`http://grinder.sourceforge.net/`

[7] Ayuda y soporte técnico de Microsoft: Herramientas de pruebas de esfuerzo para el servidor Web.
`http://support.microsoft.com/kb/231282`

[8] Balachander Krishnamurthy y Jennifer Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, 2001. ISBN 0201710889.

[9] Ricardo Cao Abad. *Introducción a la simulación y a la teoría de colas*. Netbiblo, 2002. ISBN 8497450175.

[10] World Wide Web Consortium.
`http://www.w3.org/TR/html401/frameset.dtd`

[11] Lorem Ipsum Generator.
`http://www.lorem-ipsum.info/generator3-es`

[12] Open Standards (WG14-C).
`http://www.open-std.org/JTC1/SC22/WG14/`

[13] The Single UNIX Specification, Version 3.
`http://www.unix.org/version3/`

[14] The Single UNIX Specification, Utility Conventions.
`http://www.opengroup.org/pubs/online/7908799/xbd/utilconv.html`

[15] Alonso Álvarez García y José Ángel Morales Grela. *HTML 4.1: Edición Actualizada*. Anaya Multimedia, 2003. ISBN 8441515832.

[16] James L. Antonakos y Kenneth C. Mansfield. *Programación estructurada en C*. Prentice-Hall, 1997. ISBN 8489660239.

[17] Juan Diego Gutierrez Gallardo. *Desarrollo web con PHP 5 y MySQL*. Anaya Multimedia, 2004. ISBN 8441517746.

[18] Hrair Aldermeshian y Thomas B. London. *Redes locales y seguridad digital*. Anaya Multimedia, 2003. ISBN 8441515492.

[19] William Mendenhall y Terry Sincich. *Probabilidad y estadística para ingeniería y ciencias*. Prentice Hall Hispanoamericana, 1997. ISBN 9688809608.