Unidad 4.

Evaluación de Prestaciones (Benchmarking)

Aurora Macías Ojeda

Abril de 2017

Trabajo realizado como requisito parcial para superar la asignatura Servicios de Computación de Altas Prestaciones y Disponibilidad, perteneciente al Máster Universitario en Ingeniería Informática (MUii)

Escuela Superior de Ingeniería Informática - Albacete
Universidad de Castilla-La Mancha

Índice de Contenidos

Índice de Figuras

1 Introducción

En la actualidad, determinados tipos de aplicaciones tales como aplicaciones científicas o de inteligencia de negocio, requieren una gran capacidad de procesamiento o un uso eficiente de otros recursos del sistema para llevar a cabo operaciones en un marco de tiempo razonable. Esas necesidades se abordan con la computación de altas prestaciones (*High Performance Parallel and Distributed Computing*, HPDC) por las ventajas que ésta ofrece (ver temas anteriores). Con el objetivo de lograr la configuración más apropiada para la ejecución de las aplicaciones mencionadas, es necesario evaluar las infraestructuras de altas prestaciones de acuerdo a los requisitos de dichas aplicaciones cuantificando la potencia o capacidad para llevar a cabo ciertas tareas requeridas [1], esto es, las prestaciones del sistema.

Este trabajo pretende ofrecer un resumen del estado del arte relativo a la evaluación de las prestaciones de sistemas informáticos, aunque no necesariamente de altas prestaciones. En el Apartado 2 se comentan las razones que hacen necesaria la evaluación, los objetivos que podría tener, y la metodología que habría que seguir para llevarla a cabo satisfactoriamente. En el Apartado 3 se incluyen gran parte de los grupos de parámetros que se usan para describir el comportamiento de un sistema y que constituyen los indicadores en los que se apoya la evaluación. A continuación, en el Apartado 4, se resumen las técnicas de evaluación más comunes. En el apartado 5 se comentan algunas herramientas de evaluación usadas, relacionadas con las técnicas de evaluación referenciadas. Por último, en el Apartado 6, se recogen las conclusiones del trabajo destacando que la evaluación de prestaciones de los sistemas de altas prestaciones no es un asunto que esté ya cerrado.

2 Necesidades de la Evaluación. Objetivos y Metodología

En general, es necesario evaluar un sistema informático para (i) *comprobar* o predecir que su funcionamiento es el correcto o el esperado. Pero la necesidad también existe cuando se pretende (ii) adquirir o *elegir* un equipo informático, (iii) *diseñar* un sistema, (iv) *comparar* varios sistemas que realizan tareas determinadas, (v) *planificar* la capacidad de un sistema, o (vi) sintonizar, ajustar, o *configurar* o ampliar un sistema. La evaluación se puede aplicar a todas las fases del ciclo de vida de un sistema informático [2]–[4].

Los objetivos y la metodología de evaluación de un sistema informático, ya sea de altas *prestaciones*[1] o no, se detallan en los apartados que figuran a continuación.

[1] Facilidades que un sistema de procesamiento de información es capaz de proporcionar a sus usuarios.

2.1 Objetivos de la Evaluación

La evaluación de las prestaciones de un sistema informático se lleva a cabo con algún objetivo. Entre los objetivos más comunes se encuentran los siguientes [3], [5]:

- comparar distintos sistemas sujetos a elección;
- determinar el impacto de un nuevo elemento o característica que se ha incorporado al sistema (adición de un disco duro nuevo, etc.);
- sintonizar el sistema, es decir, mejorar su funcionamiento en relación a algún aspecto determinado;
- medir prestaciones concretas y relativas de diferentes sistemas;
- *depurar prestaciones identificando los fallos del sistema que provocan lentitud del mismo* (cuellos de botella)[2]; o
- establecer expectativas sobre el uso del sistema (determinar cuántas conexiones simultáneas es capaz de soportar una base de datos, cuántas peticiones permite un sitio web, etc.).

2.2 Metodología de Evaluación

La evaluación de prestaciones de sistemas informáticos, al igual que sucede con cualquier otro proceso, ha de seguir una metodología cuyas fases o pasos son [3], [6]:

1. Determinar los objetivos de la evaluación.

   Es necesario establecer qué se va a medir, y aislar la parte del sistema en cuestión del resto de partes para evitar la influencia de los demás factores.

2. Listar los servicios que ofrece el sistema y sus posibles resultados.

   Los sistemas, o sus servicios, pueden producir resultados válidos, inválidos, o no producir ningún resultado. Se debe medir la tasa de sucesos de cada una de las opciones mencionadas.

3. Seleccionar los criterios y las métricas para comparar las prestaciones.

   La selección dependerá de la parte o de las prestaciones concretas del sistema que se quiera evaluar o medir. En base a ello, las métricas serán distintas. Las posibles métricas que podrían considerarse se verán más adelante (ver Sección 3).

4. Listar los parámetros que pueden afectar a las prestaciones.

[2] Objetivo puesto en práctica en la mayor parte de actividades de este tema.

Han de estar categorizados como (i) características del sistema y (ii) características de la *carga*[3] de trabajo a la que está sometido el sistema. Los parámetros podrán tomar distintos valores que mantendrán una correspondencia con diversos niveles establecidos.

5. Seleccionar los factores que se van a evaluar y estudiar.

   Para la selección deben considerarse todos los parámetros que pueden afectar a las prestaciones determinados previamente. Algunos de ellos son bastante susceptibles de variar durante la evaluación como sucede con los relativos a la carga de trabajo, los relacionados con las características del sistema, por el contrario, permanecen invariables ante sistemas con el mismo hardware.

6. Elegir la/s técnica/s de evaluación a usar.

   La elección deberá realizarse atendiendo a las características de las distintas técnicas que se verán más tarde: monitorización, modelado, o benchmarking (ver Sección 4).

7. Seleccionar la carga de trabajo a la que se someterá el sistema para medir sus prestaciones.

   Debería ser una carga de trabajo genérica, a la que se llega tras analizar la carga de trabajo de diversos sistemas o de un mismo sistema con diversas aplicaciones.

8. Diseñar los experimentos.

   Los experimentos estarán divididos de acuerdo a los niveles o valores que tomarán los factores considerados. Idealmente, se diseñan experimentos contemplando muchos factores pero pocos niveles al principio, y posteriormente se pone énfasis en los factores que han resultado tener más influencia en el experimento.

9. Analizar e interpretar los datos.

   La medición por sí sola no es suficiente. Además, hay que realizar una síntesis de los datos y extraer conclusiones a partir de ellos.

10. Presentar los resultados.

    La presentación de los datos es importante con independencia del objetivo de la evaluación y el destinatario de los datos a presentar. El resumen de los resultados suele presentarse gráficamente, siendo común el uso de diagramas de Gant o gráficos de Kiviat[4].

[3] Demanda de servicios ofrecidos por un sistema, realizada por los usuarios del mismo en un intervalo de tiempo determinado.

[4] Gráfico circular en el que los radios representan los índices de prestaciones.

Es necesario tener en cuenta que, llegados al punto final, podría ser necesario comenzar otra vez el proceso de evaluación, normalmente porque hayan cambiado los objetivos.

3 Parametrización de los Sistemas

Con el propósito de que la evaluación del sistema pueda satisfacer el objetivo o los objetivos con los que se realiza, el comportamiento del sistema tiene que describirse y para ello es parametrizado, esto es, se definen parámetros directamente relacionados con el comportamiento del sistema o con otros aspectos del mismo relevantes para la evaluación. Los parámetros definidos harán referencia a medidas cuantitativas enmarcadas en algunos de estos tipos [2], [5]: (i) consumo de tiempo, (ii) utilización de recursos, o (iii) trabajo realizado por el sistema o componentes del mismo.

Puesto que la evaluación se llevará a cabo sometiendo el sistema a una carga, la carga también tiene asociada una parametrización. Tanto los parámetros relativos al sistema como los de la carga que se miden normalmente, se recogen en las secciones que figuran a continuación.

No se incluyen, por contra, las magnitudes usadas para controlar el comportamiento del sistema ante modificaciones que se pueden realizar con el objetivo de mejorarlo, así como tampoco las magnitudes medibles susceptibles de ser utilizadas como índices de prestaciones basadas fundamentalmente en el tiempo en el que el sistema lleva a cabo una tarea determinada y que están directamente asociadas al procesador (MIPS, MFLOPS, etc.).

3.1 Parámetros relativos al Sistema

Los parámetros asociados al sistema propiamente dicho pueden dividirse, a su vez, en tres grupos dependiendo de si son o no perceptibles por usuarios o administradores del sistema, por ejemplo, o de si no están directamente relacionados con las prestaciones del sistema. Esa división y los parámetros o variables correspondientes se esboza en las secciones siguientes.

3.1.1 Variables Externas o Perceptibles por los Usuarios

Los indicadores que pueden ser percibidos por personas relacionadas de alguna manera con el sistema son las siguientes.

- *Productividad (throughput)*: cantidad de trabajo útil *ejecutado* por unidad de tiempo en un determinado entorno de carga. Con "trabajo" puede hacerse referencia a transacciones, procesos, instrucciones, etc. "Útil" se refiere a que el trabajo se realiza bajo la operación normal del sistema con algún propósito funcional.
- *Capacidad:* cantidad máxima de trabajo útil *realizable* por unidad de tiempo en un determinado entorno de carga (productividad máxima que puede obtenerse).

- *Tiempo de respuesta:* tiempo transcurrido entre la entrega de un trabajo al sistema y la recepción del resultado o de la respuesta (tiempo en procesar un trabajo).

3.1.2 Variables Internas o del Sistema

Las variables que habitualmente caracterizan un sistema informático constituyendo índices o indicadores internos son las recogidas a continuación.

- *Factor de utilización*: tiempo en el que un componente determinado ha sido utilizado realmente.
- *Solapamiento de componentes*: tiempo en el que varios componentes han sido utilizados simultáneamente.
- *Sobrecarga (overhead)*: carga que se ha procesado y que no ha sido solicitada explícitamente por los usuarios.
- *Factor de carga de multiprogramación*: relación entre el tiempo de respuesta de un trabajo en un entono de multiprogramación y en uno de monoprogramación.
- *Frecuencia de fallo de página*: número de fallos de página producidos por unidad de tiempo en un sistema de memoria virtual paginada.
- *Frecuencia de swapping*[5] *o intercambio*: número de programas expulsados de memoria por unidad de tiempo.

3.1.3 Otras Variables relativas al Comportamiento

A continuación, se incluyen otras variables que suelen caracterizar el comportamiento de un sistema. Es necesario destacar que estas variables son factores de calidad que pueden asociarse también a un producto software y no solamente al sistema completo.

- *Fiabilidad*: probabilidad en un intervalo de tiempo concreto de que el sistema funcione correctamente.
- *Disponibilidad*: probabilidad de que en un instante concreto el sistema esté trabajando correctamente y pueda realizar sus funciones.
- *Seguridad*: probabilidad de que el sistema, tanto si realiza correctamente sus funciones como si está detenido por algún problema, no perturbe el funcionamiento de otros sistemas ni comprometa la seguridad de las personas relacionadas con él.

[5] Mecanismo o modo de interrelación entre la memoria principal (memoria que contiene el programa en ejecución, los datos de proceso inmediato, y los resultados intermedios) con la memoria secundaria o de apoyo. Un proceso o parte de él se traslada temporalmente desde la memoria principal para ser devuelto posteriormente. El propósito de esta técnica es posibilitar que el sistema operativo pueda asignar más memoria de la que tiene realmente.

- *Rendimiento (performance)*: probabilidad de que las prestaciones del sistema estén sobre un nivel determinado en un instante de tiempo dado.
- *Mantenibilidad*: facilidad con la que un sistema estropeado pueda ser reparado y devuelto al estado operacional dentro de un periodo de tiempo determinado.

3.2 Parámetros relativos a la Carga

Elegir la carga de prueba que se usará en la evaluación de las prestaciones del sistema es importantísimo. El resultado de la evaluación se obtiene en función de la carga utilizada, y cuando se detallan índices de prestación debe especificarse también la carga con la que se obtuvieron.

La carga de prueba puede ser la *carga real* del sistema o, lo más común (sobre todo por la inflexibilidad y falta de reproducibilidad de la carga real), una carga específicamente generada (ya sea *sintética* o *artificial*, en cuyos detalles no vamos a entrar) y que en algún caso puede no ser ejecutable (algunos tipos de carga artificial). En cualquier caso, la carga de prueba (tanto si es real como si no) se debe caracterizar de acuerdo a distintas magnitudes.

Cuando la carga está formada por diversos componentes, los parámetros que habría que considerar por cada componente serían los que se indican a continuación.

- *Tiempo de la CPU por trabajo*: tiempo total de CPU necesario para ejecutar un programa, transacción, etc. en un sistema determinando. Depende directamente del número de instrucciones que se ejecutan para el trabajo en cuestión, del volumen de datos procesados, y de la velocidad del procesador.
- *Número total de operaciones de E/S por trabajo*. Conviene desglosarlo por tipo de dispositivos a los que se accede.
- *Características de las operaciones de E/S por trabajo*: soporte sobre el que se realizan, posición que ocupa el archivo sobre el que se efectúan (en caso de que el soporte fuese un disco), etc.
- *Prioridad* asignada por el usuario a cada trabajo al que se somete al sistema.
- *Memoria* necesaria para la ejecución de un trabajo determinado. Podría ser constante (memorial real) o variable (memoria virtual paginada o segmentada).
- *Localidades de las referencias* a memoria: tiempo en el que todas las referencias a memoria hechas por un trabajo permanecen dentro de una página o conjunto de páginas. A mayor tiempo dentro de la misma página, mayor localidad.

El conjunto completo de la carga tiene atribuidas las siguientes magnitudes.

- *Tiempo entre llegadas*: tiempo entre dos requerimientos sucesivos para un servicio (ejecución de un trabajo o transacción) del sistema.
- *Frecuencia de llegada* (inversa del tiempo medio entre llegadas): número medio de llegadas de nuevas peticiones de ejecución producidas por unidad de tiempo.
- *Distribución de trabajos*: proporción existente entre ejecuciones de los distintos trabajos que constituyen la carga.

Existen casos en los que hay que considerar también las acciones del usuario (cargas conversacionales) y habría que medir adicionalmente los siguientes parámetros.

- *Tiempo de reflexión del usuario*: tiempo que el usuario de un terminal de un sistema interactivo necesita para generar una nueva petición al sistema. En ese tiempo, el usuario lee la respuesta del sistema y piensa la siguiente acción que va a realizar de acuerdo a la respuesta recibida.
- *Número de usuarios simultáneos* de un sistema interactivo en un instante dado.
- *Intensidad del usuario*: relación entre el tiempo de respuesta de una petición y el tiempo de reflexión del usuario.

4 Técnicas de Evaluación

En alguna de las fases de la metodología esquematizada anteriormente, se hacía referencia a la elección de técnicas para llevar a cabo la evaluación. Entre las técnicas usadas más habitualmente para evaluar sistemas informáticos se encuentran la (i) monitorización o medición, el (ii) modelado, y el (iii) benchmarking [2].

4.1 Monitorización

La monitorización consiste en la medición de las prestaciones de un sistema informático determinado en funcionamiento, realizando un seguimiento de su comportamiento. La monitorización resulta imprescindible para evaluar el comportamiento, sin embargo, puede introducir perturbaciones en el sistema a evaluar.

La monitorización se lleva a cabo mediante el uso de herramientas conocidas como monitores. Los monitores pueden ser software, hardware (dispositivos electrónicos conectados a los sistemas, ej.: sondas para medir temperatura de los equipos) o híbridos (software que capta acontecimientos en el sistema operativo y los envía a un módulo hardware conectado al bus del sistema) [6]. Una descripción más detallada de algunos de ellos será proporcionada más adelante (ver Sección 5.1).

4.2 Modelos

Existen ocasiones en las que se pretende evaluar un sistema que no está en ejecución. En esos casos, suele ser común el uso de las técnicas de modelado analítico o simulación descritas a continuación.

4.2.1 Modelado Analítico

Los modelos analíticos se usan para evaluar sistemas incompletos o no construidos aún. Esos modelos son representaciones formales del sistema construidos usando fórmulas y ecuaciones diferenciales (modelos matemáticos). Con ellos se trata de hallar, a partir de valores conocidos o estimados de ciertas variables o parámetros, los valores para esos parámetros que van a resultar de interés. El término "analítico" hace referencia a la técnica utilizada para resolver el modelo: resolución mediante fórmulas cerradas o algoritmos aproximados de las ecuaciones matemáticas que representan el equilibrio existente entre transiciones de estados o eventos producidos en el sistema [6].

En general, el fundamento se halla en la teoría de colas. "Cola" hace referencia a una línea de espera. La teoría de colas (o de líneas de espera) es aplicable a problemas caracterizables como de congestión llegada-partida, esto es, aquellos en los que un conjunto de clientes llega a un centro de servicio para ser atendido y el servidor, que dispone de cierta capacidad de atención, no está disponible. Entonces, si el cliente decide esperar, se forma una cola. Las colas pueden considerarse individuales (ver Figura 1, a) o bien unidas formando redes; las redes pueden ser tanto abiertas (ver Figura 1, b) como cerradas (ver Figura 1, c) [4], [6].

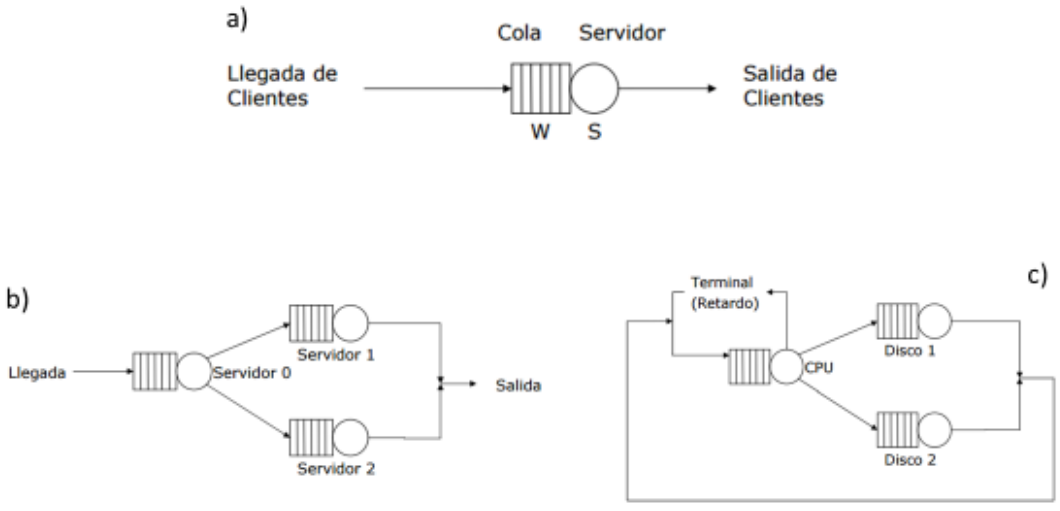

Figura 1. Esquema de componentes y organizaciones de una cola (extraído de [4]*).*

De manera general, los trabajos en los ordenadores comparten recursos. En un instante de tiempo determinado solamente un trabajo puede utilizar un recurso. El resto han de esperar en

una cola para utilizarlo. Lo que se pretende determinar en estos casos sería, sobretodo, el tiempo de espera.

Es necesario tener en cuenta que los métodos analíticos son incapaces de tratar determinadas estructuras y comportamientos de colas existentes en los sistemas informáticos.

4.2.2 Simulación

La simulación también se basa en modelos matemáticos, en este caso numéricos. Consiste en el desarrollo de un programa que reproduce el comportamiento temporal del sistema en base a sus estados y transiciones. Los resultados corresponden a la extracción de estadísticas del comportamiento simulado del sistema.

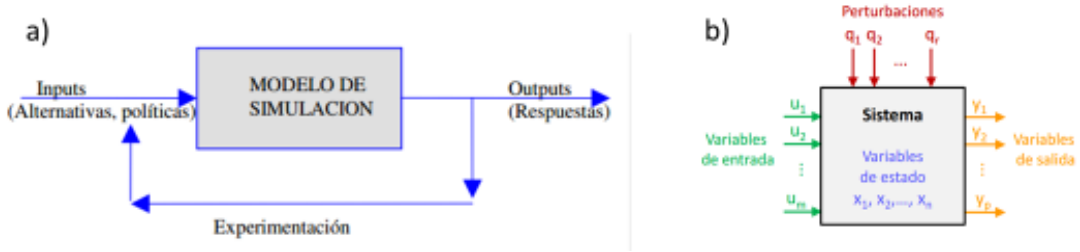

Figura 2. Esquema del proceso experimental de la simulación (a) (extraído de ([6])), y detalle del modelo del sistema que se somete a simulación (b).

La simulación, por sus características y por los desarrollos computacionales de los últimos años, presenta una serie de ventajas que la convierten en el procedimiento de evaluación más adecuado en muchos casos, y en algunos la única alternativa [6].

4.3 Benchmarking

La evaluación de sistemas informáticos suele realizarse normalmente ejecutando determinadas aplicaciones de prueba conocidas como *benchmarks (suites)*.

En general, un programa o paquete de programas benchmark evalúa diferentes aspectos de las prestaciones de un sistema informático, o de una parte del mismo, reproduciendo en él una carga genérica de trabajo. Al proceso de comparar dos o más sistemas por la obtención de medidas se le denomina *benchmarking*.

Los benchmarks poseen un cierto grado de oficialidad o estandarización. No obstante, cada benchmark tiene especificaciones y reglas para la ejecución diferentes a las del resto.

Los paquetes benchmark y las cargas de trabajo de prueba deben reunir una serie de características para cumplir su función adecuadamente [3].

- *Reproductibilidad*: tanto el programa en sí como los resultados a lo largo de diferentes ejecuciones deben poder reproducirse fácilmente para cualquier tipo de sistema.

- *Compacidad*: debe tener poco código, no debe tardar demasiado tiempo en ejecutarse, y no debe sobrecargar el sistema.
- *Compatibilidad*: la carga de prueba debe ser compatible con todos los sistemas susceptibles de ser medidos (es la característica más importante).
- *Resolución*: el benchmark debe durar lo suficiente como para que sea fácil diferenciar entre dos sistemas con características similares.
- *Escalabilidad*: el mismo benchmark deber servir para una amplia gama de velocidades, debe seguir siendo útil a lo largo del tiempo, y debe dar resultados significativos, esto es, que sirvan para diferenciar las prestaciones de los sistemas evaluados.

Existen diversos *tipos* de benchmarks según su forma de implementación (aplicaciones, conjunto de instrucciones mezcladas asociadas a su frecuencia de uso, etc.), así como distintos *niveles* dependiendo de si evalúan componentes del sistema o el rendimiento global del mismo.

5 Herramientas de Análisis de Rendimiento

De acuerdo a la metodología esquematizada previamente (ver Sección 2.2), tras decidir qué técnica de evaluación aplicar, habría que seleccionar también la herramienta o herramientas que se utilizarán para analizar el rendimiento del sistema.

Cada una de las técnicas que se incluyen en la sección anterior tiene asociados determinados tipos de herramientas. A continuación, se comentan algunos de los tipos más comunes y se recogen ejemplos de algunas de ellas consideradas especialmente relevantes [4], [6]–[8].

5.1 Herramientas para Monitorización

Lo denominados monitores software son los que resultan de mayor interés en este caso. En general, los monitores software pueden acceder a datos de más alto nivel y suelen ser más económicos que el resto.

Normalmente se distinguen dos tipos de monitores software [4], [6]:

- *Profilers*. Trozos de código adheridos a un programa que son llamados cada cierto tiempo, y que generan un fichero que es posteriormente analizado. El análisis muestra el tiempo empleado en cada uno de las funciones de un programa, el número de veces que han sido llamadas, etc., con el objetivo de que el programador pueda optimizar esas funciones. Suele ser necesario indicar al compilador que incluya la opción de profiling (perfilado).

  Algunos ejemplos serían Gprof, Cprof, o Valgrind.

- *Programas de medición del estado* de un sistema.

  TORQUE estaría mejor encuadrado en este otro tipo de monitores software (aunque no exactamente).

En las secciones siguientes se proporcionan más detalles sobre los ejemplos de monitores software indicados.

5.1.1 Gprof

Gprof [9] es una herramienta enmarcada dentro de los profilers que usa un híbrido de *instrumentación*[6] y muestreo. Básicamente, lo que hace es recopilar estadísticas de ejecución de los programas analizados. Hay que tener en cuenta que si durante la ejecución no se usan determinadas funciones de las que han sido programadas, no se generará información de perfil para esas funciones o características.

Gprof se ejecuta en sistemas Unix/Linux y es capaz de analizar programas escritos en C y C++.

El perfilado consta de varios pasos:

- Compilar y "linkar" (*link*)[7] el programa habilitando el perfilado.
- Ejecutar el programa para generar un archivo de datos de perfil.
- Ejecutar Gprof para analizar los datos de perfil. Para ello se usa el patrón

  gprof opciones [archivoEjecutable [archivosDatosPerfiles…]] [> archivoSalida].

Puesto que los datos por sí mismos no dicen nada, hay disponibles tres formas de salida para realizar el análisis:

- *Flat profile*: muestra cuánto tiempo emplea el programa en cada función y cuántas veces se llama a las funciones.
- *Call graph*: muestra, para cada función, qué otras funciones la llaman y cuántas veces. También proporciona una estimación sobre cuánto tiempo ha sido empleado en las subrutinas de cada función. Lo anterior podría sugerir lugares en los que se debería eliminar llamadas a las funciones que emplean mucho tiempo.

[6] Inserción de sondas (software) en el código fuente (o en el binario) que recogen información interesante: timestamps absolutos o de ejecución, información de la aplicación como pid, etc. Los datos recogidos se almacenan en bruto en un buffer y se analizan externamente.

[7] Montar o enlazar todos los archivos generados en el compilado de un programa para obtener el archivo ejecutable del mismo.

- *Annotated source listing*: copia del código fuente del programa, etiquetado con el número de veces que cada línea del programa ha sido ejecutada.

5.1.2 Cprof

Cprof [10] es otra herramienta de perfilado para programas escritos en C/C++ que funciona bajo Linux.

Cprof se usa para determinar el número de veces que son ejecutadas las diferentes funciones que componen un programa. Para ello usa puntos de ruptura (*breakpoints*) de manera similar a como se hace en la traza de llamada local (*local call trace*), pero conteniendo contadores para recopilar los datos de perfilado. Ello hace que no sea necesario compilar de manera especial ningún módulo que vaya a ser perfilado.

El perfilado con Cprof se realiza siguiendo los pasos indicados a continuación. El patrón de instrucción correspondiente a cada uno de ellos se indica en cursiva. La cardinalidad especificada corresponde al número de parámetros que es posible pasar a la función.

- Comenzar el perfilado con los contadores de llamada puestos a 0 para las funciones especificadas, estableciendo puntos de ruptura de conteo de llamadas sobre ellas.

  cprof:start/0..3

- Ejecutar el código que será perfilado.

  Mod:Fun()

- Pausar los contadores de llamada para funciones específicas. Ello minimiza el impacto en el código que se ejecuta en segundo plano (*background*) o en el *shell*[8] y que interrumpe o perturba el perfilado. Los contadores de llamada se pausan automáticamente cuando alcanzan el tope del tamaño de palabra de la máquina *host* (para una máquina con un tamaño de palabra de 32 bits, el valor máximo del contador es 2^{32-1} = 2147483647).

  cprof:pause/0..3

- Recopilar los contadores de llamada y computar el resultado.

  cprof:analyse/0..2

[8] Interfaz de usuario para acceder a los servicios del Sistema operativo. En general, los shells de los sistemas operativos usan o una interfaz de línea de comandos (CLI) o una interfaz gráfica de usuario (GUI), dependiendo del rol de del ordenador y de la operación particular. Se denomina shell porque es una capa sobre el núcleo (*kernel*) del sistema operativo.

- Reiniciar los contadores de llamada para funciones específicas. Ello puede servir para recopilar un nuevo conjunto de contadores sin tener que detener e iniciar el perfilado de conteo de llamada.

  cprof:restart/0..3

- Detener el perfilado eliminando los puntos de ruptura del conteo de llamada de funciones específicas.

  cprof:stop/0..3

Cprof puede usarse para perfilar trabajo en segundo plano, un módulo, o el código. Finalmente, el trazado de conteo de llamadas que ofrece Cprof es más liviano que otras formas de trazado puesto que no tienen que generarse mensajes de traza [11].

5.1.3 Valgrind

La suite *Valgrind* [12] es un marco de trabajo de instrumentación que constituye un conjunto de herramientas para llevar a cabo tareas de diversos tipos tales como depuración, perfilado (profiling), o similares (detección de errores de memoria, de errores en hilos, etc.) y que ayudan a los desarrolladores a mejorar los programas propios. También es posible usar Valgrind para construir nuevas herramientas de análisis dinámico.

La suite puede ejecutarse en sistemas Linux, Solaris, Android, y Mac OS (entre otros similares), y puede usarse en programas escritos en C, C++, Java, Perl, Python. ensamblador, Fortran, Ada, y muchos otros. Valgrind es apropiado para múltiples tipos de aplicaciones software: aplicaciones de escritorio, bases de datos, navegadores, software empotrado, aplicaciones de inteligencia de negocio, juegos, frameworks de realidad virtual, etc.

De manera estándar, Valgrind incorpora nueve herramientas consideradas útiles (*Memcheck*, *Cachegrind*, *Callgrind*, *Helgring*, *DRD*, *Massif*, *DHAT*, *SGcheck*, y *BBV*) y alguna más no tan importante. De entre esas herramientas se destacan las que se indican a continuación:

- *Memcheck*: detector de errores de memoria. Ayuda a los programadores a hacer sus programas más correctos (particularmente los escritos en C y C++). Es la herramienta por defecto de Valgrind.

- *Massif*: perfilador de saltos. Ayuda al programador a conseguir que sus programas usen menos memoria.

Para usar alguna de las herramientas de Valgrind, es necesario conocer las características del *core* y de las herramientas que se usarán.

De manera general, se invoca a Valgrind usando el esquema:

valgrind [opcionesValgrind] nuestroPrograma [opcionesNuestroPrograma].

La opción más importante de Valgrind es *--tool*, que permite especificar cuál de las herramientas se pretende ejecutar. El patrón de uso sería:

--tool=<nombreHerramienta> [default: memcheck].

Si esta opción no es especificada, se ejecuta la herramienta por defecto.

Con independencia de la herramienta usada, Valgrind toma el control de las aplicaciones antes de su comienzo, y dichas aplicaciones son ejecutadas en una CPU "sintética" brindada por el core. En líneas generales, el flujo sería:

1. El core de Valgrind pasa el código nuevo que queremos analizar a la herramienta seleccionada.
2. La herramienta añade su propio código de instrumentación y devuelve el resultado al core.
3. El core coordina la ejecución del código "instrumentado" (código original al que se ha añadido código de instrumentación).

5.1.4 Tera-scale Open-source Research and QUEue manager (TORQUE)

La herramienta conocida como *TORQUE* [13] es un gestor de recursos distribuido que provee control sobre *trabajos batch* (*batch jobs*)[9] y nodos de computación distribuidos.

La topología de TORQUE y los tres componentes de servicio en los que se basa (*pbs_server, pbs_mon,* y *trqauthd*) no se verán debido al carácter introductorio del tema.

La ejecución de un job usando TORQUE involucra los siguientes pasos:

1. Crear un script de envío de job (archivo con extensión .pbs o .sh, o sin extensión).
2. Enviar el job. Se obtiene un número ID de envío (submission ID number).

   $ qsub <scriptEnvioJob>
3. Esperar hasta que la ejecución del job haya terminado. Mientras se lleva a cabo la ejecución, es posible (i) consultar el estado del job -o (ii) de todos los jobs en la cola- o (iii) eliminar el envío.

   (i) *$ qstat <jobID>*

[9] Programa de ordenador o conjunto de programas de ordenador procesados en modo batch. Dicho modo implica que una secuencia de comandos a ejecutar por el sistema operativo sea listada en un archivo (denominado archivo batch, archivo de comandos, o shell script) y enviado para su ejecución como una unidad. Lo contrario de un job batch es el procesamiento interactivo, consistente en la inserción de comandos individuales por parte del usuario para que sean procesados inmediatamente.

(ii) *$ qstat [<jobID>]*

(iii) *$ qdel <jobID>*

4. Recuperar la salida o los resultados del job.

TORQUE cuenta con otros comandos además de los indicados, y es posible usar opciones que extienden sus posibilidades. Algunos ejemplos son el envío de secuencias o arrays de jobs (opción *-t* de *qsub*) o la ejecución condicional de jobs (opción *-W* de *qsub*).

Es común encontrar que el uso de TORQUE para registrar y monitorizar se apoye en planificadores de job adicionales como *Moab*, un gestor de carga de trabajo, para manejar y finalizar (*dispatch*) los jobs en base a la disponibilidad de los recursos de computación, requisitos especificados por los usuarios, y políticas de uso establecidas por los administradores del clúster.

5.2 Herramientas para Modelado

Existen diversas herramientas (y lenguajes) para evaluar el rendimiento de un sistema basándose en algún tipo de técnica de modelado [14]. No obstante, *Matlab* y *Simulink* [15] son las herramientas utilizadas para modelado y simulación (en este segundo caso conjuntamente) en prácticamente todos los contextos actuales.

5.3 Herramientas para Benchmarking

Las herramientas benchmark están divididas en grupos según la carga de trabajo que evalúan. En algunos casos los benchmark están incluso propuestos por los propios fabricantes de sistemas [3], [7].

Algunos ejemplos de bechmarks populares son los benchmarks *SPEC* (Systems Performance Evaluation Cooperative) [16], que constituyen paquetes para evaluar el rendimiento de la CPU fundamentalmente; *Linpack,* muy usado en entornos científicos por su capacidad de resolución de ecuaciones lineales y que tiene asociada la implementación *HPL* [17] para sistemas de memoria distribuida; o *HPC Challenge* [18], que combina diversas suites de benchmarking para medir el rendimiento de sistemas HPC (*High Performace Computing*) en base a cuatro atributos importantes: aritmética de punto flotante y doble precisión, ancho de banda de memoria local, ancho de banda de red para mensajes “grandes”, ancho de banda de red para mensajes “pequeños”. Esta suite de benchmark incluye a HPL. Además, tiene asociada la *HPC Challenge Award Competition*, que pretende promover el desarrollo de suites HPC que contribuyan al uso productivo de los sistemas HPC.

6 Conclusiones

Los sistemas de computación de altas prestaciones tienen un número creciente de aplicaciones y también de uso y, al igual que ocurre con el resto de sistemas informáticos, es importante evaluar sus prestaciones para determinar que cumplen con los requisitos esperados.

La evaluación de prestaciones de sistemas informáticos en general se lleva a cabo siguiendo una metodología que incluye la selección de características a evaluar, de técnicas de evaluación, y de herramientas asociadas a ellas de acuerdo a las necesidades y a los objetivos de la evaluación a llevar a cabo.

Este trabajo recoge un resumen de los aspectos de la metodología de evaluación destacados y proporciona, también, ejemplos de herramientas de evaluación usadas.

Es importante destacar que aún hay desafíos en la evaluación de sistemas informáticos en general, y de altas prestaciones en particular, y que incluso existen premios para promover el desarrollo de nuevas y mejores herramientas de evaluación.

7 Referencias

[1] M. Ferro, A. R. Mury, L. F. Manfroi, and B. Schlze, "High Performance Computing Evaluation A methodology based on Scientific Application Requirements," *eprint arXiv:1412.1297*, 2014. .

[2] I. Gil, "Análisis y Diseño de Sistemas Informáticos. Apuntes," 2014. [Online]. Available: http://personales.upv.es/igil/adsi.pdf. [Accessed: 23-Mar-2017].

[3] J. J. Merelo, "Diseño y Evaluación de Configuraciones. Apuntes," 2003. [Online]. Available: http://kal-el.ugr.es/~jmerelo/DyEC/. [Accessed: 23-Mar-2017].

[4] M. Á. Villarroel, "Evaluación y Explotación de Sistemas Informáticos. Apuntes," 2011. [Online]. Available: https://www2.infor.uva.es/~miguelv/eesi/. [Accessed: 23-Mar-2017].

[5] B. B. Vergara, "Análisis Comparativo del Rendimiento," 2011. [Online]. Available: http://exa.unne.edu.ar/informatica/SO/Analisis Comparativo del Rendimiento.pdf. [Accessed: 23-Mar-2017].

[6] Ó. A. Vallejos, "Evaluación de Sistemas y Procesamiento de Datos. Apuntes," *2012*. [Online]. Available: http://exa.unne.edu.ar/informatica/evalua_ant/. [Accessed: 23-Mar-2017].

[7] L. K. John, "Performance Evaluation: Techniques, Tools and Benchmarks," 2010. [Online]. Available: https://lca.ece.utexas.edu/pubs/john_perfeval.pdf. [Accessed: 23-Mar-2017].

[8] G. Casale, M. Gribaudo, and G. Serazzi, "Tools for Performance Evaluation of Computer Systems: Historical Evolution and Perspectives," in *Performance Evaluation of Computer and Communication Systems. Milestones and Future Challenges: IFIP WG 6.3/7.3 International Workshop, PERFORM 2010, in Honor of G{ü}nter Haring on the Occasion of His Emeritus Celebration, Vienna, Austria, October 14-16,* , K. A. Hummel, H. Hlavacs, and W. Gansterer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 24–37.

[9] University of California, “Gprof,” 2008. [Online]. Available: https://sourceware.org/binutils/docs/gprof/. [Accessed: 23-Mar-2017].

[10] A. Lewycky, “Cprof,” 2003. [Online]. Available: http://cprof.sourceforge.net/. [Accessed: 23-Mar-2017].

[11] Ericsson AB, “cprof - The Call Count Profiler,” 2017. [Online]. Available: http://erlang.org/doc/apps/tools/cprof_chapter.html. [Accessed: 01-Apr-2017].

[12] Valgrind™ Developers, “Valgrind,” 2016. [Online]. Available: http://valgrind.org/. [Accessed: 22-Mar-2017].

[13] I. Adaptive Computing, “TORQUE Resource Manager,” 2017. [Online]. Available: http://www.adaptivecomputing.com/products/open-source/torque/. [Accessed: 23-Mar-2017].

[14] B. R. Haverkort and I. G. Niemegeers, “Performability modelling tools and techniques,” *Perform. Eval.*, vol. 25, no. 1, pp. 17–40, 1996.

[15] I. The MathWorks, “Simulink,” 2016. [Online]. Available: https://es.mathworks.com/products/simulink.html. [Accessed: 23-Mar-2017].

[16] Standard Performance Evaluation Corporation, “SPEC’s Benchmarks,” 2017. [Online]. Available: https://www.spec.org/benchmarks.html. [Accessed: 23-Mar-2017].

[17] University of Tennessee, “HPL Benchmark.” [Online]. Available: http://www.netlib.org/benchmark/hpl/. [Accessed: 23-Mar-2017].

[18] University of Tennessee, “HPC Challenge,” 2017. [Online]. Available: http://icl.cs.utk.edu/hpcc/. [Accessed: 23-Mar-2017].