

ISE 22/23 - Tema 4: Análisis comparativo del rendimiento

¿Qué servidor tiene mejor rendimiento?

Objetivos:

- *Entender la problemática inherente al diseño de un índice de rendimiento cualquiera.*
- *Interpretar los índices clásicos de rendimiento usados en el ámbito de los procesadores.*
- *Entender el concepto de benchmark y sus distintos tipos.*
- *Conocer ejemplos reales de benchmarks*
- *Conocer diferentes estrategias de análisis para hacer comparaciones de rendimiento así como las condiciones para hacer una comparación de rendimiento lo más ecuánime posible.*

Características de un buen índice de rendimiento de un sistema informático

- Repetibilidad: Siempre que se mida el índice en las mismas condiciones, el valor de éste debe ser el mismo.
- Representatividad y fiabilidad: Si un sistema A siempre presenta un índice de rendimiento mejor que el sistema B, es porque siempre el rendimiento real de A es mejor que el de B.
- Consistencia: El índice se debe poder medir en cualquier sistema informático, independientemente de su arquitectura o de su S.O.
- Facilidad de medición: Sería deseable que las medidas necesarias para obtener el índice sean fáciles de tomar.
- Linealidad: Sería deseable que, si el índice de rendimiento aumenta, el rendimiento real del sistema aumente en la misma proporción.

Algunos índices de rendimiento: ¿valen la pena?

Tiempo de ejecución, frecuencia de reloj y CPI

¿Pueden ser la frecuencia de reloj (f_{RELOJ}) o el número medio de ciclos por instrucción (CPI) buenos índices de rendimiento?

$$T_{EJEC} = NI \times CPI \times T_{RELOJ} = \frac{NI \times CPI}{f_{RELOJ}}$$

- T_{EJEC} = Tiempo de ejecución del programa.
- NI = Número de instrucciones del programa.

No lo son. Es posible encontrar ejemplos de sistemas con f_{RELOJ} (o CPI) peores que otros, pero con mejores prestaciones. Además, CPI depende del programa que se ejecute.

¿Y si usamos directamente el tiempo de ejecución (T_{EJEC}) de un determinado programa?

- ¿Consistencia? El programa debería estar descrito en un lenguaje de alto nivel. Ok.
- ¿Repetibilidad? El programa debería ejecutarse en un entorno muy controlado. Ok.
- ¿Representatividad y fiabilidad? ¡¡¡Dependería del programa a ejecutar!!!

MIPS (millones de instrucciones por segundo)

En principio, parece una medida prometedora ya que representa cómo de rápido ejecuta las instrucciones un microprocesador.

$$MIPS = \frac{NI}{T_{EJEC} \times 10^6} = \frac{f_{RELOJ}}{CPI \times 10^6}$$

Inconvenientes:

- ¿Repetibilidad? Los MIPS medidos varían incluso entre diferentes programas en el mismo computador.
- ¿Representatividad y fiabilidad? Depende del juego de instrucciones (ej. RISC vs CISC).

MFLOPS (millones de operaciones en coma flotante por segundo)

Basado en operaciones y no en instrucciones.

$$MFLOPS = \frac{\text{Operaciones de coma flotante realizadas}}{T_{EJEC} \times 10^6}$$

Inconvenientes:

- No todas las operaciones de coma flotante tienen la misma complejidad.

- Posible solución: MFLOPS normalizados: Cada operación se multiplica por un peso que es proporcional a su complejidad.

Ejemplo de asignación de pesos:

- ADD, SUB, COMPARE, MULT \Rightarrow 1 operación normalizada
- DIVIDE, SQRT \Rightarrow 4 operaciones normalizadas
- EXP, SIN, ATAN, ... \Rightarrow 8 operaciones normalizadas
- ¿Consistencia? El formato de los números en coma flotante puede variar de una arquitectura a otra y, por tanto, los resultados de las operaciones podrían tener diferente exactitud.
- Además, ¿y si no son importantes las operaciones en coma flotante en mi servidor?

En conclusión: no nos vale, ni tampoco ninguno de los índices anteriores, no quedan candidatos. Nos contentaremos con el tiempo de ejecución (TEJEC) de un determinado programa o conjunto de programas

El índice de rendimiento va a depender de la carga que se use para hacer la comparación.

Carga real

Difícil de utilizar para la comparación de rendimiento de un servidor por las siguientes razones:

- Varía a lo largo del tiempo: ¿Uso la carga de hoy, la de ayer...?
- Resulta complicado reproducirla: Si la uso con un servidor muy lento, ¿es posible que llegue una solicitud basada en una respuesta que todavía no ha llegado!!
- Interacciona con el sistema informático: Si un servidor contesta muy rápido, es posible que reciba las solicitudes antes o incluso más solicitudes que si fuera muy lento.



Es más conveniente utilizar un modelo de la carga real como carga de prueba (test workload) para hacer comparaciones.

Modelo de carga

Representatividad

Los modelos de carga son representaciones aproximadas de la carga que recibe un sistema informático. El modelo de la carga:

- Debe ser lo más representativo posible de la carga real.
- Debe ser lo más simple/compacto que sea posible (tiempos de medición y espacio en memoria razonables).



Estrategias para obtener modelos de carga

- Ajustar un modelo paramétrico “personalizado” a partir de la monitorización del sistema ante la carga real (**caracterización de la carga**).

La forma más fácil para obtener un modelo de la carga a la que está sometido un servidor durante un determinado periodo de tiempo consiste en:

- Identificar los recursos que más demande la carga (CPU, memoria, discos, red, etc.)
- Elegir los parámetros característicos de dichos recursos (utilización de CPU, lecturas/escrituras que hay que hacer en cada disco, lecturas/escrituras a memoria, número de accesos a la red, etc.)
- Medir el valor de dichos parámetros usando monitores de actividad (muestreo).
- Analizar los datos: medias, histogramas, agrupamiento o clustering, etc.
- Generar el modelo de carga seleccionando representantes de la carga (=solicitudes al servidor) junto con información estadística sobre su distribución temporal.

Ventajas:

- La carga de prueba es muy representativa de la carga real.

Desventajas:

- El proceso de obtención de la carga de prueba es bastante tedioso.

- Desconocemos el rendimiento de otros servidores distintos al nuestro con esa carga de prueba.
- Usar cargas de prueba que usen un modelo genérico de carga lo más similar posible al que se quiere reproducir (**referenciación o benchmarking**).

4.1. Referenciación (benchmarking)

Consiste en utilizar un programa o un conjunto de programas (benchmark programs) estándar con el fin de comparar alguna característica del rendimiento (también llamada “métrica”) entre equipos informáticos. Hay dos características principales que definen a un benchmark:

- La carga de prueba (test workload) específica con la que estresa el sistema evaluado.
- El conjunto de reglas que se deben seguir para la correcta ejecución, obtención y validación de los resultados.

Algunas ventajas de usar benchmarking:

- Existen muchos benchmarks diferentes para distintos tipos de servidores y cargas. Hay una alta probabilidad de encontrar uno que reproduzca unas condiciones parecidas a las que experimenta nuestro servidor.
- Las comparaciones entre el rendimiento de varios servidores son justas ya que todas las ejecuciones se realizan de forma idéntica siguiendo las reglas del benchmark.
- Muchos benchmarks permiten ajustar la carga de tal forma que podemos medir la escalabilidad de nuestro servidor.
- Al poder conocer tanto el rendimiento para un determinado benchmark que obtienen diferentes servidores como cómo están diseñados y configurados dichos servidores, obtenemos una información muy valiosa sobre cómo diseñar y/o configurar nuestros propios servidores.

Tipos de programas benchmark

Según la estrategia de medida

- Programas que miden el tiempo necesario para ejecutar una cantidad preestablecida de tareas.
 - La mayoría de benchmarks.
- Programas que miden la cantidad de tareas ejecutadas para un tiempo de cómputo pre-establecido.

- SLALOM: Mide la exactitud de la solución de un determinado problema que se puede alcanzar en 1 minuto de ejecución.
- TPC-C: Calcula cuántas consultas por segundo se realizan, de media, a un servidor de base de datos permitiendo aumentar tanto el nº de usuarios como el tamaño de la base de datos. Exige un tiempo mínimo de respuesta para un determinado tanto por ciento de peticiones.

Según la generalidad del test

- Microbenchmarks o benchmarks para componentes: estresan componentes o agrupaciones de componentes concretos del sistema: procesador, caché, memoria, discos, red, procesador+caché, procesador+compilador+memoria virtual, etc.
- Macrobenchmarks o benchmarks de sistema completo o de aplicación real: la carga intenta imitar situaciones reales (normalmente servidores con muchos clientes) típicas de algún área. P.ej. comercio electrónico, servidores web, servidores de ficheros, servidores de bases de datos, sistemas de ayuda a la decisión, paquetes ofimáticos + correo electrónico + navegación, etc.

Ejemplos de microbenchmarks

- Whetstone (1976) – Mide el rendimiento de las operaciones en coma flotante por medio de pequeñas aplicaciones científicas que usan sumas, multiplicaciones y funciones trigonométricas.
- Linpack (1983) – Mide el rendimiento de las operaciones en coma flotante a través de un algoritmo para resolver un sistema denso de ecuaciones lineales. El benchmark incorpora una rutina para comprobar que la solución a la que se llega es la correcta con un grado de exactitud prefijado. Se utiliza para confeccionar la lista de los 500 mejores supercomputadores del mundo.
- Dhrystone (1984) – Mide el rendimiento de operaciones con enteros, esencialmente por medio de operaciones de copia y comparación de cadenas de caracteres.
- Stream: para medir el ancho de banda de las memoria DRAM y las cachés del microprocesador (SRAM).
- IOzone: rendimiento del sistema de ficheros (p.ej. lecturas y escrituras a/desde el disco duro). Igualmente HD Tune (Windows), Iometer, fio (Linux) o el programa 'hdparm -tT' (Linux).
- Iperf: rendimiento TCP y UDP (Linux y Windows). Debe estar instalado tanto en el cliente como en el servidor. O el programa pchar (Linux), calcula el ancho de banda por cada salto de IP hasta alcanzar el destino.

- También existen aplicaciones que incorporan varios paquetes de microbenchmarks para poder realizar diversos tests de forma cómoda:
 - AIDA64 (Windows)
 - Sandra (Windows)
 - Phoronix Test Suite (Multiplataforma, código abierto)
 - Permite la instalación, ejecución y la generación de informes tanto de tests de rendimiento (test profiles) como de agrupaciones de éstos (test suites) de forma automatizada.
 - Permite recopilar automáticamente en tiempo de ejecución la información de sensores del sistema (consumo de energía, temperaturas, voltajes, frecuencias CPU, velocidad ventiladores, utilización de CPU/GPU/discos/memoria, etc.)
 - Permite visualizar el resultado de forma gráfica en html y/o exportarlo a <https://openbenchmarking.org> para compartirlo con otros y/o compararlo con otros resultados.

El paquete de microbenchmarks SPEC CPU 2017

SPEC (Standard Performance Evaluation Corporation), es una corporación sin ánimo de lucro cuyo propósito es establecer, mantener y respaldar la estandarización de benchmarks y herramientas para evaluar el rendimiento y la eficiencia energética de los equipos informáticos.

Algunos miembros de la corporación: Acer, AMD, Amazon Web Services, Apple, Google, HP, IBM, Intel, Microsoft, Oracle...

Compuesto por cuatro conjuntos de microbenchmarks distintos:

- Speed: Cuánto tarda en ejecutarse un programa (tiempo de respuesta)
 - SPECspeed®2017 Integer (rendimiento en aritmética entera)
 - SPECspeed®2017 Floating Point (rendimiento en coma flotante)
- Rate: Cuántos programas puedo ejecutar por unidad de tiempo (productividad)
 - SPECrate®2017 Integer (rendimiento en aritmética entera)
 - SPECrate®2017 Floating Point (rendimiento en coma flotante)

SPEC CPU2017 se distribuye como una imagen ISO que contiene:

- Código fuente de todos los programas de benchmark.
- Data sets que necesitan algunos benchmarks para su ejecución.

- Herramientas varias para compilación, ejecución, obtención de resultados, validación y generación de informes.
- Documentación, incluyendo reglas de ejecución y de generación de informes.

El tiempo de ejecución depende del índice a obtener, la máquina en la que se ejecuta y cuántas copias o subprocesos se eligen.

Los programas dentro del benchmark SPECspeed 2017 deben ser aplicaciones reales portables a múltiples arquitecturas. Algunos son:

- SPECspeed®2017 Integer – 10 programas, mayoría en C/C++
 - Intérprete de Perl
 - Utilidad de compresión
 - Compilador de C
 - Conversión XML a HTML
- SPECspeed®2017 Floating Point – 10 programas, la mayoría en Fortran/C
 - Dinámica de fluidos
 - Predicción meteorológica
 - Procesamiento de imágenes

Los índices de prestaciones de SPEC, también llamados genéricamente, índices SPEC son:

- Aritmética entera: CPU2017IntegerSpeed__peak, CPU2017IntegerSpeed__base.
- Aritmética en coma flotante: CPU2017FP__Speed__peak, CPU2017FP__Speed__base.

El significado de base y peak en los índices SPEC:

- Base: Compilación en modo conservador: todos los programas escritos en el mismo lenguaje usan las mismas opciones de compilación.
- Peak: Rendimiento pico, permitiendo que cada uno escoja las opciones de compilación óptimas para cada programa.

Cada programa del benchmark se ejecuta 3 veces y se escoge el resultado intermedio (se descartan los 2 extremos). El índice SPEC es la media geométrica de las ganancias en velocidad con respecto a una máquina de referencia (en SPEC CPU2017 una Sun Fire V490).

Ejemplo: Si llamamos t_i al tiempo que tarda la máquina a evaluar en ejecutar el programa de benchmark i -ésimo y t_{iREF} lo que tardaría la máquina de referencia para ese programa (y hay 10 programas en el benchmark):

$$\text{índice SPEC} = \sqrt[10]{\frac{t_{1}^{REF}}{t_1} \times \frac{t_{2}^{REF}}{t_2} \times \dots \times \frac{t_{10}^{REF}}{t_{10}}}$$

Resultados de SPECspeed® 2017 Integer



All SPEC CPU2017 Integer Speed Results Published by SPEC

These results have been submitted to SPEC; see the [disclaimer](#) before studying any results.

[Search published CPU2017 results](#)

Last update: 2017-10-19T11:49

CPU2017 Integer Speed (7):

[\(Search in CPU2017 Integer Speed results\)](#)

Test Sponsor	System Name	Parallel	Base Threads	Processor				Results	
				Enabled Cores	Enabled Chips	Threads/Cores	Base	Peak	
HPE	Integrity Superdome X (384 core, 2.20 GHz, Intel Xeon E7-8890 v4) HTML CSV Test PDF PS Config	No	384	384	16	2	5.31	5.86	
HPE	ProLiant DL580 Gen9 (2.20 GHz, Intel Xeon E7-8890 v4) HTML CSV Test PDF PS Config	No	96	96	4	1	5.35	5.95	
HPE	ProLiant ML350 Gen9 (2.20 GHz, Intel Xeon E5-2699 v4) HTML CSV Test PDF PS Config	No	44	44	2	1	5.80	6.43	
HPE	ProLiant DL380 Gen10 (2.10 GHz, Intel Xeon Platinum 8170) HTML CSV Test PDF PS Config	Yes	52	52	2	1	8.96	Not Run	
HPE	ProLiant DL380 Gen10 (2.10 GHz, Intel Xeon Platinum 8176) HTML CSV Test PDF PS Config	Yes	56	56	2	1	9.16	Not Run	
Huawei	Huawei 2288H V5 (Intel Xeon Platinum 8180) HTML CSV Test PDF PS Config	Yes	56	56	2	1	9.46	9.79	
Oracle Corporation	Sun Fire V490 HTML CSV Test PDF PS Config	Yes	1	8	4	1	1.00	Not Run	

Resultados de SPECspeed® 2017 Integer (II)

Hardware				Software			
CPU Name:	Intel Xeon E7-8890 v4			OS:	SUSE Linux Enterprise Server 12 (x86_64) SP1		
Max MHz:	3400				3.12.53-60.30-default		
Nominal:	2200			Compiler:	C/C++: Version 17.0.0.098 of Intel C/C++		
Enabled:	384 cores, 16 chips, 2 threads/core				Compiler for Linux;		
Orderable:	2 to 16 chips				Fortran: Version 17.0.0.098 of Intel Fortran		
Cache L1:	32 KB I + 32 KB D on chip per core			Compiler for Linux:			
L2:	256 KB I+D on chip per core			Parallel:	No		
L3:	60 MB I+D on chip per chip			Firmware:	HP Bundle: 008.004.084 SPFW: 043.025.000 08/16/2016		
Other:	None			File System:	xfs		
Memory:	4 TB (128 x 32 GB 2Rx4 PC4-2400T.L, running at 1600 MHz)			System State:	Run level 5 (multi-user, w/GUI)		
Storage:	8 x C8S59A, 900 GB 10 K RPM SAS			Base Pointers:	64-bit		
Other:	None			Peak Pointers:	32/64-bit		
				Other:	Microquill SmartHeap V10.2		

Results Table																
Benchmark	Base								Peak							
	Threads	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Threads	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio		
600.perfbench_s	384	365	4.86	358	4.96	357	4.98	384	296	5.95	295	6.02	295	6.01		
602.gcc_s	384	553	7.20	546	7.29	546	7.29	384	540	7.37	538	7.48	534	7.45		
605.mcf_s	384	866	5.45	866	5.45	898	5.26	384	708	6.67	700	6.75	699	6.73		
620.omnettp_s	384	276	5.90	271	6.03	289	5.65	384	251	6.50	247	6.61	246	6.64		
623.xalancbench_s	384	189	7.50	188	7.52	187	7.57	384	179	7.21	179	7.23	180	7.87		
625.x264_s	384	283	6.24	282	6.25	283	6.23	384	271	6.51	272	6.49	270	6.52		
631.deepsjeng_s	384	407	3.52	408	3.52	407	3.52	384	343	4.18	343	4.18	343	4.18		
640.taoiba_s	384	460	3.64	460	3.63	460	3.63	384	436	3.90	430	3.98	440	3.93		

Resultados de SPECspeed® 2017 Integer (III)

Base Optimization Flags	
C benchmarks:	<code>-qopt-prefetch -qopt-mem-layout-trans=3 -DSPEC_SUPPRESS_OPENMP</code>
C++ benchmarks:	<code>-Wl,-zmuldefs -qopt-prefetch -qopt-mem-layout-trans=3 -DSPEC_SUPPRESS_OPENMP -Lshlib10.2 -lsmathexp64</code>
Peak Optimization Flags	
C benchmarks:	<p>600.perlbench_s: <code>-prof-gen(pass 1) -prof-use(pass 2) -O2 -xCORE-AVX2 -auto-p32 -ipo -qopt-prefetch -O3 -no-prec-div -qopt-mem-layout-trans=3 -DSPEC_SUPPRESS_OPENMP</code></p> <p>602.qcc_s: Same as 600.perlbench_s</p> <p>605.mcf_s: <code>-prof-gen(pass 1) -prof-use(pass 2) -ipo -xCORE-AVX2 -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 -DSPEC_SUPPRESS_OPENMP</code></p> <p>625.x264_s: Same as 600.perlbench_s</p> <p>657.xz_s: Same as 600.perlbench_s</p>
C++ benchmarks:	<p>620.omnetpp_s: <code>-Wl,-zmuldefs -prof-gen(pass 1) -prof-use(pass 2) -ipo -xCORE-AVX2 -O3 -no-prec-div -auto-p32 -qopt-prefetch -qopt-mem-layout-trans=3 -DSPEC_SUPPRESS_OPENMP -Lshlib10.2 -lsmathexp64</code></p> <p>623.xalancbmk_s: Same as 620.omnetpp_s</p> <p>631.deepsjeng_s: <code>-Wl,-zmuldefs -prof-gen(pass 1) -prof-use(pass 2) -ipo -xCORE-AVX2 -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 -DSPEC_SUPPRESS_OPENMP -Lshlib10.2 -lsmathexp64</code></p> <p>641.leela_s: Same as 620.omnetpp_s</p>

Ejemplo de cálculo de SPECspeed® 2017 Integer_{base}

Benchmark	t ^{REF} (s)	Exp1 (s)	Exp2 (s)	Exp3 (s)	t _{base} (s)	t ^{REF} / t _{base}
600.perlbench_s	1774	365	358	357	358	4,96
602.qcc_s	3981	553	546	546	546	7,29
605.mcf_s	4721	866	866	898	866	5,45
620.omnetpp_s	1630	276	271	289	276	5,91
623.xalancbmk_s	1417	189	188	187	188	7,54
625.x264_s	1764	283	282	283	283	6,23
631.deepsjeng_s	1432	407	408	407	407	3,52
641.leela_s	1706	469	469	469	469	3,64
648.exchange2_s	2939	329	329	329	329	8,93
657.xz_s	6182	2165	2161	2164	2164	2,86

$$\text{SPECspeed@2017 Integer}_{\text{base}} = \sqrt[10]{\frac{t_1^{\text{REF}}}{t_1^{\text{base}}} \times \frac{t_2^{\text{REF}}}{t_2^{\text{base}}} \times \dots \times \frac{t_{10}^{\text{REF}}}{t_{10}^{\text{base}}}} = \sqrt[10]{4,96 \times 7,29 \times 5,45 \times \dots} = 5,31$$

Ejemplos de benchmarks de sistema completo: TPC

TPC (Transactions Processing Performance Council). Organización sin ánimo de lucro especializada en benchmarks relacionados con servidores que procesan datos (data-driven servers). Empresas que hay detrás de TPC: AMD, Cisco, Dell, Fujitsu, HP, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle...

Principales tipos de benchmarks:

- OLTP (on-line transaction processing): Se trata de realizar pequeñas transacciones (compras, consultas,...) con unos requisitos exigentes de tiempos de respuesta.

- DS (decision support): Se trata de realizar consultas complejas que suelen requerir acceso a buena parte de una gran base de datos y un procesamiento posterior.
- IoT (Internet of Things): Se trata de procesar una gran cantidad de datos procedentes de una gran cantidad de dispositivos diferentes.
- Big Data: Se trata de procesar una gran cantidad de información usando clusters de computadores y Apache Hadoop.
- Artificial Intelligence: Se trata de ejecutar algoritmos de aprendizaje utilizados en machine learning (p.ej. redes neuronales profundas) a partir de datos.
- Virtualization: Se trata de ejecutar los benchmarks de los tipos anteriores en un entorno virtualizado en lugar de en una máquina “física” (bare metal).

EJ. Benchmark TPC-C: Es de tipo OLTP (on-line transaction processing). Simula una gran compañía que vende 100.000 productos y que tiene varios almacenes (configurable), cada uno a cargo de 10 zonas, con 3000 clientes/zona.

Las peticiones involucran acceso a las bases de datos tanto locales como distribuidas (a veces el producto no está en el almacén más cercano), ejecución simultánea de consultas y acceso no uniforme a las bases de datos.

- Índice de rendimiento utilizado: transacciones procesadas por segundo, minuto u hora (tps, tpm o tph) superando unos ciertos requisitos de tiempos de respuesta (ej. el 90% deben tener un tiempo de respuesta inferior a 5s). También suelen proporcionar tanto el consumo de potencia como el coste por transacción procesada (incluido mantenimiento de 3 años).

EJ. Benchmark TPC-H: Es de tipo DS (decision support). Contiene un total de 22 tipos de consultas diferentes que requieren examinar grandes volúmenes de datos para poder contestar a preguntas complejas. También incluyen modificaciones concurrentes de los datos. Algunos de estos tipos de consultas son:

- Realizar un informe detallado sobre las ventas en un determinado periodo de tiempo.
- Buscar el proveedor más adecuado según un conjunto determinado de criterios.
- Cuantificar el aumento de ingresos que habría resultado de eliminar ciertos descuentos en toda la empresa en un rango de fechas determinado.

Aunque es escalable, la base de datos cuenta con la información de un mínimo de 10000 proveedores con un mínimo de 10 millones de filas.

- Índice de rendimiento utilizado (métrica): TPC-H Composite Query-per-Hour Performance Metric (QphH@Size), donde Size es el tamaño de la base de datos utilizado (factor de escala). Tiene en cuenta la productividad (consultas por unidad de tiempo) en dos escenarios diferentes: cuando las consultas se realizan por un único usuario y cuando se permite concurrencia entre varios usuarios. También suelen proporcionar tanto el consumo de potencia como el coste por QphH@Size (incluido mantenimiento de 3 años).

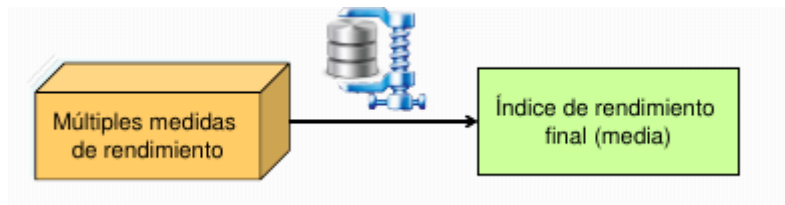
Otros benchmark de sistema completo

- Servidores de ficheros:
 - SPECstorage Solution 2020.
 - SPEC SFS2014.
- JAVA Cliente/Servidor:
 - SPECjEnterprise2010: Java Enterprise Edition (JEE).
 - SPECjms2007: Java Message Service (JMS). ◻ SPECjvm2008: Java Runtime Environment (JRE).
- High Performance Computing, OpenMP, MPI, OpenCL:
 - SPEC MPI2007: Message Passing Interface (MPI).
 - SPEC OMP2012: Open MultiProcessing (OpenMP).
 - SPEC ACCEL: OpenCL y OpenACC.
- Para el mundo del PC:
 - SPECworkstation® 3.1: Sistemas basados en Windows. Usa programas de código abierto.
 - SYSmark25: De la empresa BAPco. Sistemas basados en Windows. Usa programas propietarios.

4.2. Análisis de los resultados de un test de rendimiento

¿Cómo expresar el resultado final tras la ejecución de un test de rendimiento?

Muchos tests de rendimiento se basan en la ejecución de diferentes programas y, por tanto, producen diferentes medidas de rendimiento. Sin embargo, estos tests suelen resumir todas estas medidas en un único valor: el índice de rendimiento de dicho test.



¿Cómo concentrar todos los índices en uno solo? Método habitual de síntesis: uso de algún tipo de media.

Media aritmética

Dado un conjunto de n medidas t_1, \dots, t_n , definimos su media aritmética como el sumatorio de $t_1..t_n$ entre n :

$$\bar{t} = \frac{1}{n} \sum_{k=1}^n t_k$$

Si no todas las medidas tienen la misma importancia, se puede asociar a cada medida t_k un peso w_k , obteniéndose la media aritmética ponderada:

$$\bar{t}_W = \sum_{k=1}^n w_k \times t_k \quad \text{con} \quad \sum_{k=1}^n w_k = 1$$

Si t_k es el tiempo de ejecución del programa de benchmark k -ésimo en la máquina a testar, w_k podría escogerse, por ejemplo, inversamente proporcional a dicho tiempo de ejecución en una determinada máquina de referencia:

$$w_k \equiv \frac{C}{t_k^{REF}} \quad \rightarrow \quad C = \frac{1}{\sum_{k=1}^n 1/t_k^{REF}}$$

Media geométrica

Dado un conjunto de n medidas, S_1, \dots, S_n , definimos su media geométrica como la raíz n -ésima del multiplicatorio de $S_1..S_n$:

$$\bar{S}_g = \sqrt[n]{\prod_{k=1}^n S_k} = \left(\prod_{k=1}^n S_k \right)^{1/n}$$

Propiedad: cuando las medidas son ganancias en velocidad (speedups) con respecto a una máquina de referencia, este índice mantiene el mismo orden en las comparaciones independientemente de la máquina de referencia elegida (siempre que sea la misma). Usado en los benchmarks de SPEC y SYSmark25.

$$SPEC(M) = \frac{\sqrt[n]{t_1^{REF} \times t_2^{REF} \times \dots \times t_n^{REF}}}{\sqrt[n]{t_1^M \times t_2^M \times \dots \times t_n^M}} = \frac{\sqrt[n]{t_1^{REF} \times t_2^{REF} \times \dots \times t_n^{REF}}}{\sqrt[n]{t_1^M \times t_2^M \times \dots \times t_n^M}}$$

$$SPEC(M1) > SPEC(M2) \Leftrightarrow \sqrt[n]{t_1^{M1} \times t_2^{M1} \times \dots \times t_n^{M1}} < \sqrt[n]{t_1^{M2} \times t_2^{M2} \times \dots \times t_n^{M2}}$$

Ejemplo de comparación con tiempos

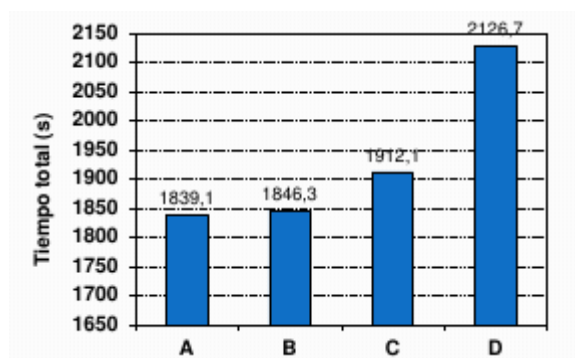
Programa	t^{REF} (s)	t^A (s)	t^B (s)	t^C (s)	t^D (s)
1	1400	141	170	136	134
2	1400	154	166	215	25
3	1100	96,8	94,2	146	201
4	1800	271	283	428	523
5	1000	83,8	90,1	77,4	81,2
6	1200	179	189	199	245
7	1300	120	131	87,7	75,5
8	300	151	158	138	192
9	1100	93,5	122	88	118
10	1900	133	173	118	142
11	1500	173	170	179	240
12	3000	243	100	100	150
Suma	17000	1839,1	1846,3	1912,1	2126,7

La máquina más rápida es “A” ya que es la que tarda menos en ejecutar, uno tras otro, todos los programas del benchmark (1839,1 segundos).

Comparación según tiempo total

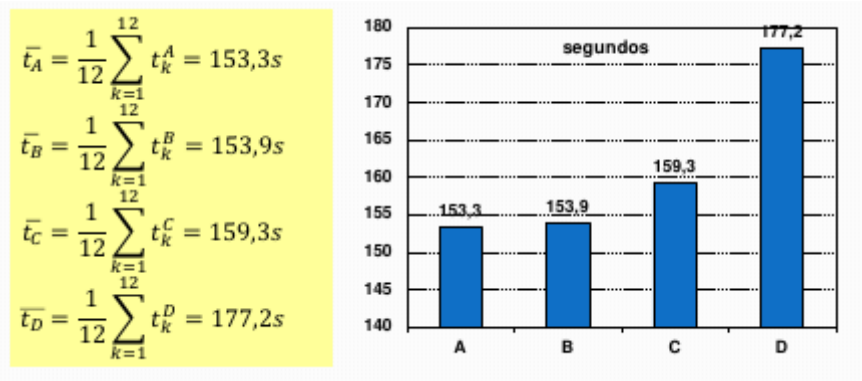
Ordenación con el tiempo total:

- De más rápida a más lenta: A, B, C, D
- Esto no significa que A sea siempre la más rápida (depende del programa), aunque, en conjunto, sí que lo es.



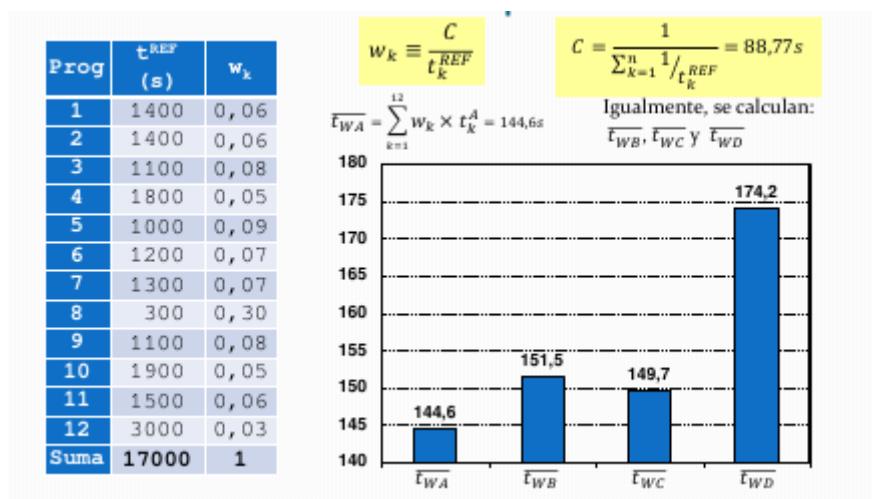
Comparación según media aritmética

La máquina que ejecuta los programas del benchmark, uno tras otro, en menor tiempo es la de menor media aritmética de los tiempos de ejecución.



Comparación según media aritmética ponderada

Según este criterio, la máquina “más rápida” sería la de menor tiempo medio ponderado de ejecución. Nótese que esta ponderación depende, en este ejemplo, de la máquina de referencia.



Comparación según media geométrica

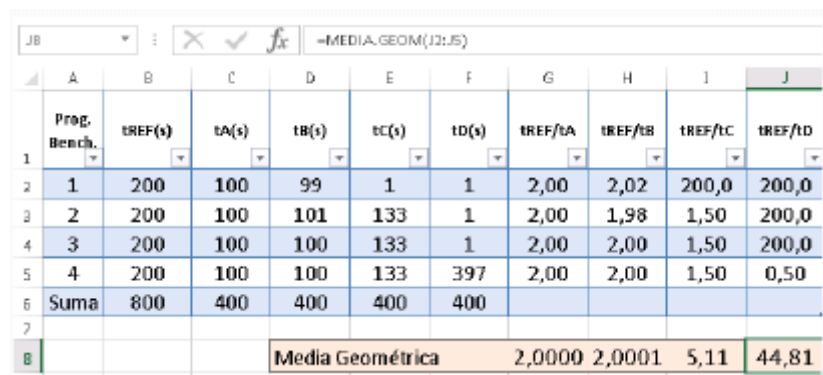
Calculamos la ganancia en velocidad de cada máquina con respecto a la máquina de referencia (tal y como lo hacen SPEC y SYSmark25):

Programa	t_{REF} (s)	S^A speedup	S^B speedup	S^C speedup	S^D speedup
1	1400	9,9	8,2	10,3	10,4
2	1400	9,1	8,4	6,5	56,0
3	1100	11,4	11,7	7,5	5,5
4	1800	6,6	6,4	4,2	3,4
5	1000	11,9	11,1	12,9	12,3
6	1200	6,7	6,3	6,0	4,9
7	1300	10,8	9,9	14,8	17,2
8	300	2,0	1,9	2,2	1,6
9	1100	11,8	9,0	12,5	9,3
10	1900	14,3	11,0	16,1	13,4
11	1500	8,7	8,8	8,4	6,3
12	3000	12,3	30,0	30,0	20,0
M. Geom.		8,78	8,66	8,97	9,00

El speedup es un índice a maximizar. Según este índice, la “mejor máquina” es ¡¡¡la D!!!

¿A quién beneficia la decisión de usar la media geométrica de speedups?

Se premian las mejoras sustanciales. No se castigan empeoramientos no tan sustanciales. Debemos ser MUY cuidadosos con las comparaciones y saber qué estamos haciendo realmente.



The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	J
1	Prog. Bench.	tREF(s)	tA(s)	tB(s)	tC(s)	tD(s)	tREF/tA	tREF/tB	tREF/tC	tREF/tD
2	1	200	100	99	1	1	2,00	2,02	200,0	200,0
3	2	200	100	101	133	1	2,00	1,98	1,50	200,0
4	3	200	100	100	133	1	2,00	2,00	1,50	200,0
5	4	200	100	100	133	397	2,00	2,00	1,50	0,50
6	Suma	800	400	400	400	400				
7										
8				Media Geométrica			2,0000	2,0001	5,11	44,81

Conclusiones de este análisis

Intentar reducir un conjunto de medidas de un test de rendimiento a un solo “valor medio” final no es una tarea trivial.

La media aritmética de los tiempos de ejecución es una medida fácilmente interpretable e independiente de ninguna máquina de referencia. El menor valor nos indica la máquina que ha ejecutado el conjunto de programas del test, uno tras otro, en un tiempo menor.

La media aritmética ponderada nos permite asignar más peso a algunos programas que a otros. Esa ponderación debería realizarse, idealmente, según las necesidades del usuario. Si se hace de forma dependiente de los tiempos de ejecución de una máquina de referencia, la elección de ésta puede influir significativamente en los resultados.

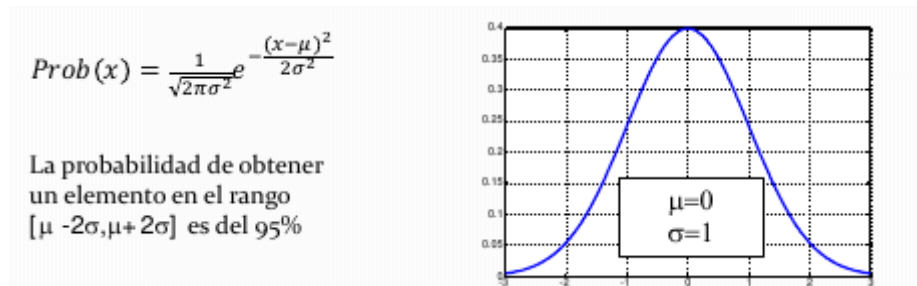
La media geométrica de las ganancias en velocidad con respecto a una máquina de referencia es un índice de interpretación compleja cuya comparación no depende de la máquina de referencia. Premia mejoras sustanciales con respecto a algún programa del test y no castiga al mismo nivel los empeoramientos.

4.3. Comparación de prestaciones en presencia de aleatoriedad

Repaso de estadística

Independientemente de qué índice se escoja, un buen ingeniero debería, en primer lugar, determinar si las diferencias entre las medidas obtenidas por un test de rendimiento en presencia de aleatoriedad son estadísticamente significativas. Necesitaremos repasar algunos conceptos de estadística.

- **Distribución normal:** Es una distribución de probabilidad caracterizada por su media μ y su varianza σ^2 cuya función de probabilidad viene dada por:



- **Teorema del límite central:** la media de un conjunto grande de muestras aleatorias de cualquier distribución e independientes entre sí pertenece a una distribución normal.
- Si extraemos N muestras $\{d_1..d_N\}$ pertenecientes a una distr normal, cuya media es μ , y calculo la siguiente media:

$$t_{exp} = \frac{\bar{d} - \bar{d}_{real}}{s/\sqrt{n}}$$

siendo \bar{d} la media muestral y s la desviación típica muestral

$$\bar{d} = \frac{\sum_{i=1}^n d_i}{n} \quad s = \sqrt{\frac{\sum_{i=1}^n (d_i - \bar{d})^2}{n-1}}$$

$s/\sqrt{n} \equiv \text{Error estándar}$

y repito muchas veces, veremos que los t_{exp} obtenidos pertenecen a una distribución t de estuden con N-1 grados de libertad (df, degrees of freedom)

Ejemplo

Tenemos los tiempos de ejecución en segundos de 6 programas P1..P6 en dos máquinas diferentes, A y B, en condiciones en las que puede haber aleatoriedad.

Programa	t_A (s)	t_B (s)	$d = t_A - t_B$ (s)
P1	142	100	42
P2	139	92	47
P3	152	128	24
P4	112	82	30
P5	156	148	8
P6	166	171	-5

Las medias de t_A y t_B son 144.5s y 120.2s, la tercera columna D es la diferencia entre las dos, y su media es 24.3s

La desviación típica (fórmula arriba, nos la suelen dar σ) es de 19.9s, así que el error estándar es $19.9/\sqrt{6}=8.12s$

¿Es una diferencia significativa entre las dos máquinas, o sólo la aleatoriedad haciendo de las suyas?

Si partimos de la hipótesis "nula" H_0 de que las máquinas tienen rendimientos equivalentes, entonces las diferencias D se producen enteramente por una suma de elementos aleatorios independientes. En tal caso, D_i serían muestras de una distribución normal [Teorema del límite central] de media 0 ($\mu_{Dreal} = 0$).

Por tanto:

$$t_{exp} = \frac{\bar{d}}{s/\sqrt{n}} = \frac{24,3s}{8,12s} = 2,99$$

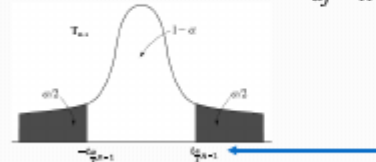
pertenecerá a una distribución t de Student con $N-1 = 6-1 = 5$ grados de libertad. **¿Qué probabilidad hay de que esto sea realmente así?** Vamos a verlo con el grado de significatividad

- **P value:** la probabilidad de obtener un valor cuyo valor absoluto es 2.99 (T_{exp}) en una distr TStudent con 5 grados de libertad (de -5 a +5) es de 0.03 [nos darían el valor o la tabla]
 - ¿Esto es mucho o poco? Definimos un umbral, el **grado de significatividad α** . También está el **grado de confianza**, $1-\alpha$ (y nos queda en forma de %).
 - Un valor típico para α es 0.05 así que tomamos eso. Grado de confianza entonces 95%.
- ENTONCES, si **pvalue** < α ($0.03 < 0.05$) diremos que, para un grado de significatividad α o para un grado de confianza del 95%, las máquinas tienen rendimientos distintos, o sea, DESCARTAMOS LA H_0 DE QUE SON IGUALES.
 - En caso contrario, simplemente NO PODEMOS DESCARTAR LA H_0 . Ni se confirma ni se desmiente.
- Así que, sabiendo esto, podemos hacer la proporción: tiempo medio a/tiempo medio b = $144.5 / 120.2 = 1.2$, la máquina B es un 20% más rápida que la máquina A

Resumen: Test t para muestras pareadas

• Partimos de:

Exp.	tA	tB	$d_i = tA_i - tB_i$
P_1	tA_1	tB_1	d_1
P_2	tA_2	tB_2	d_2
...
P_n	tA_n	tB_n	d_n



$$df = n-1$$

α

α	0.05	0.01	0.05	0.01	0.05	0.01
1	1.960	2.576	1.960	2.576	1.960	2.576
2	1.959	2.576	1.959	2.576	1.959	2.576
3	1.958	2.576	1.958	2.576	1.958	2.576
4	1.957	2.576	1.957	2.576	1.957	2.576
5	1.956	2.576	1.956	2.576	1.956	2.576
6	1.955	2.576	1.955	2.576	1.955	2.576
7	1.954	2.576	1.954	2.576	1.954	2.576
8	1.953	2.576	1.953	2.576	1.953	2.576
9	1.952	2.576	1.952	2.576	1.952	2.576
10	1.951	2.576	1.951	2.576	1.951	2.576
11	1.950	2.576	1.950	2.576	1.950	2.576
12	1.949	2.576	1.949	2.576	1.949	2.576
13	1.948	2.576	1.948	2.576	1.948	2.576
14	1.947	2.576	1.947	2.576	1.947	2.576
15	1.946	2.576	1.946	2.576	1.946	2.576
16	1.945	2.576	1.945	2.576	1.945	2.576
17	1.944	2.576	1.944	2.576	1.944	2.576
18	1.943	2.576	1.943	2.576	1.943	2.576
19	1.942	2.576	1.942	2.576	1.942	2.576
20	1.941	2.576	1.941	2.576	1.941	2.576
21	1.940	2.576	1.940	2.576	1.940	2.576
22	1.939	2.576	1.939	2.576	1.939	2.576
23	1.938	2.576	1.938	2.576	1.938	2.576
24	1.937	2.576	1.937	2.576	1.937	2.576
25	1.936	2.576	1.936	2.576	1.936	2.576
26	1.935	2.576	1.935	2.576	1.935	2.576
27	1.934	2.576	1.934	2.576	1.934	2.576
28	1.933	2.576	1.933	2.576	1.933	2.576
29	1.932	2.576	1.932	2.576	1.932	2.576
30	1.931	2.576	1.931	2.576	1.931	2.576
31	1.930	2.576	1.930	2.576	1.930	2.576
32	1.929	2.576	1.929	2.576	1.929	2.576
33	1.928	2.576	1.928	2.576	1.928	2.576
34	1.927	2.576	1.927	2.576	1.927	2.576
35	1.926	2.576	1.926	2.576	1.926	2.576
36	1.925	2.576	1.925	2.576	1.925	2.576
37	1.924	2.576	1.924	2.576	1.924	2.576
38	1.923	2.576	1.923	2.576	1.923	2.576
39	1.922	2.576	1.922	2.576	1.922	2.576
40	1.921	2.576	1.921	2.576	1.921	2.576
41	1.920	2.576	1.920	2.576	1.920	2.576
42	1.919	2.576	1.919	2.576	1.919	2.576
43	1.918	2.576	1.918	2.576	1.918	2.576
44	1.917	2.576	1.917	2.576	1.917	2.576
45	1.916	2.576	1.916	2.576	1.916	2.576
46	1.915	2.576	1.915	2.576	1.915	2.576
47	1.914	2.576	1.914	2.576	1.914	2.576
48	1.913	2.576	1.913	2.576	1.913	2.576
49	1.912	2.576	1.912	2.576	1.912	2.576
50	1.911	2.576	1.911	2.576	1.911	2.576
51	1.910	2.576	1.910	2.576	1.910	2.576
52	1.909	2.576	1.909	2.576	1.909	2.576
53	1.908	2.576	1.908	2.576	1.908	2.576
54	1.907	2.576	1.907	2.576	1.907	2.576
55	1.906	2.576	1.906	2.576	1.906	2.576
56	1.905	2.576	1.905	2.576	1.905	2.576
57	1.904	2.576	1.904	2.576	1.904	2.576
58	1.903	2.576	1.903	2.576	1.903	2.576
59	1.902	2.576	1.902	2.576	1.902	2.576
60	1.901	2.576	1.901	2.576	1.901	2.576

- H_0 : Rendimiento A \equiv Rendimiento B, es decir, $d_i \sim \mathcal{N}(\bar{d}_{real}, \sigma^2)$ con $\bar{d}_{real} = 0$
- Cálculo $t_{exp} = \frac{\bar{d} - \bar{d}_{real}}{s/\sqrt{n}} \sim T_{n-1}$ siendo $\bar{d} = \frac{\sum_{i=1}^n d_i}{n}$ $s = \sqrt{\frac{\sum_{i=1}^n (d_i - \bar{d})^2}{n-1}}$
- Definimos el nivel o grado de significatividad α .
- Rechazamos H_0 para un nivel de confianza $(1 - \alpha) * 100(\%)$ si:
 1. Método 1: $p\text{-value} < \alpha$. Siendo $p\text{-value} = P(|t| \geq |t_{exp}|)$ en $T_{n-1} \approx \text{Prob}(\text{Ho podría ser cierta})$.
 2. Método 2: $t_{exp} \notin [-t_{\alpha/2, n-1}, t_{\alpha/2, n-1}]$. Siendo $t_{\alpha/2, n-1}$ el valor que hace que $\text{Prob}(|t| > t_{\alpha/2, n-1}) = \alpha$ para una distribución t de Student con n-1 grados de libertad.
 3. Método 3: $0 \notin \left[\bar{d} - \frac{s}{\sqrt{n}} \times t_{\alpha/2, n-1}, \bar{d} + \frac{s}{\sqrt{n}} \times t_{\alpha/2, n-1} \right]$. Intervalo de confianza para \bar{d}_{real} .

56

4.4. Diseño de experimentos de comparación de rendimiento

En resumen: Test ANOVA de un factor

#Experimento	Rendimiento nivel 1	Rendimiento nivel 2	...	Rendimiento nivel n_{niv}
1	y_{11}	y_{12}		$y_{1n_{niv}}$
2	y_{21}	y_{22}		$y_{2n_{niv}}$
...
n_{rep}	$y_{n_{rep}1}$	$y_{n_{rep}2}$		$y_{n_{rep}n_{niv}}$

- **Modelo:** $y_{ij} = m_{global} + \epsilon_j + r_{ij}$ $i=1, \dots, n_{rep}; j=1, \dots, n_{niv}$
- H_0 : Rendimiento de todos los niveles del factor es equivalente ($\epsilon_j=0, j=1, \dots, n_{niv}$) = El factor no influye en el rendimiento.
- Se calcula $F_{exp} \equiv \frac{SSA/(n_{niv}-1)}{SSE/(n_{niv} \times (n_{rep}-1))} \sim F_{n_{niv}-1, n_{niv} \times (n_{rep}-1)}$
- $p\text{-value} \approx \text{Prob}(\text{Ho podría ser cierta})$.
- Rechazamos H_0 para un nivel de confianza $(1 - \alpha) * 100(\%)$ si $\text{valor-p} < \alpha$. En ese caso, el siguiente paso será comparar las medias de cada nivel unas con otras usando un test t: *prueba de múltiples rangos o de comparaciones múltiples*.