A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

19-12-2016

# Memoria Práctica 4

Ingeniería de Servidores

Several thin, curved lines in shades of blue and grey originate from the bottom left and sweep upwards and to the right.

Andrés Molina López  
UNIVERSIDAD DE GRANADA

## **Índice:**

1. Seleccione, instale y ejecute uno, comente los resultados.....	3
2. De los parámetros que le podemos pasar al comando ¿Qué significa -c 5? ¿y -n 100? Monitorice la ejecución de ab contra alguna máquina (cualquiera) ¿cuántas “tarefas” crea ab en el cliente?.....	5
3. Ejecute ab contra las tres máquinas virtuales (desde el SO anfitrión a las máquinas virtuales de la red local) una a una (arrancadas por separado). ¿Cuál es la que proporciona mejores resultados? Muestre y coméntelos. (Use como máquina de referencia Ubuntu Server para la comparativa).....	6
4. Instale y siga el tutorial en <a href="http://jmeter.apache.org/usermanual/build-web-test-plan.html">http://jmeter.apache.org/usermanual/build-web-test-plan.html</a> realizando capturas de pantalla y comentándolas. En vez de usar la web de jmeter, haga el experimento usando sus máquinas virtuales ¿coincide con los resultados de ab?.....	8
5. Programe un benchmark usando el lenguaje que desee. El benchmark debe incluir: .....	11
1) Objetivo del benchmark.	
2) Métricas (unidades, variables, puntuaciones, etc.).	
3) Instrucciones para su uso.	
4) Ejemplo de uso analizando los resultados.	
6. Bibliografía .....	22

## **Índice de figuras:**

1ª Ilustración.....	3
2ª Ilustración.....	4
3ª Ilustración.....	4
4ª Ilustración.....	5
5ª Ilustración.....	5
6ª Ilustración.....	6
7ª Ilustración.....	7
8ª Ilustración.....	8
9ª Ilustración.....	9

10ª Ilustración.....	9
11ª Ilustración.....	9
12ª Ilustración.....	10
13ª Ilustración.....	10
14ª Ilustración.....	10
15ª Ilustración.....	11
16ª Ilustración.....	13
17ª Ilustración.....	13
18ª Ilustración.....	14

## **1ª Cuestión: Seleccione, instale y ejecute uno, comente los resultados.**

El benchmark que he elegido entre todos los disponibles es systemd-boot-total, el cual mide el tiempo total de arranque del sistema. La prueba la he hecho en CentOS 7, de modo que para instalar Phoronix Suite hay que ejecutar los siguientes comandos una vez descargado el comprimido de su página oficial:

```
tar -xzf phoronix-test-suite-[versión que tengamos].tar.gz
```

```
cd phoronix-test-suite
```

```
./install.sh
```

Una vez instalado podemos ejecutar la siguiente orden para ver los benchmarks disponibles en la suite:

```
/usr/bin/phoronix-test-suite list-tests
```

En esta lista se puede encontrar sin problemas el benchmark mencionado anteriormente, del cual podemos encontrar más información en la página de openBenchmarking [1].

Para instalarlo ejecutamos el siguiente comando:

```
phoronix-test-suite install pts/systemd-boot-total
```

Y, por último, para ejecutar el benchmark escribimos el siguiente comando:

```
phoronix-test-suite benchmark pts/systemd-boot-total
```

Cuando lo ejecutamos vemos la pantalla que muestro en la *Ilustración 1*, en la que se nos deja probar varias opciones distintas, para hacer la prueba más completa, le daremos al 6, que comprueba todas las opciones.

```
[AndMolLop 21/12/16 ~]# phoronix-test-suite benchmark pts/systemd-boot-total
Phoronix Test Suite v6.8.0

Installed: pts/systemd-boot-total-1.0.4

Systemd Total Boot Time:
pts/systemd-boot-total-1.0.4
Processor Test Configuration
  1: Total
  2: Userspace
  3: Kernel
  4: Loader
  5: Firmware
  6: Test All Options
Test: █
```

*Ilustración 1. Opciones disponibles de systemd-boot-total*

Al ejecutarse lo primero que se muestra es la información del sistema tanto de hardware como de software, como se ve en la *Ilustración 2*.

## System Information

### Hardware:

Processor: Intel Core i7-4700HQ @ 2.39GHz (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: Intel 440FX- 82441FX PMC, Memory: 1024MB, Disk: 2 x 11GB VBOX HDD, Graphics: LLVMpipe, Audio: Intel 82801AA AC 97 Audio, Network: Intel 82540EM Gigabit

### Software:

OS: CentOS Linux 7, Kernel: 3.10.0-514.2.2.el7.x86\_64 (x86\_64), Desktop: GNOME Shell 3.14.4, Display Server: X Server 1.17.2, Display Driver: modesetting 1.17.2, OpenGL: 2.1 Mesa 11.2.2 Gallium 0.4, File-System: ext4, Screen Resolution: 1024x768, System Layer: KVM VirtualBox

Would you like to save these test results (Y/n): █

Ilustración 2. Información del sistema a nivel hardware y software

Al continuar con la ejecución comprueba las 5 opciones mostradas anteriormente, y nos las va mostrando una por una. Si en la *Ilustración 2* le dijimos que sí en la opción de guardar, cuando termine de comprobar las distintas opciones, nos permitirá abrir los resultados en el navegador.

Si lo abrimos en el navegador lo primero que se nos muestra es el informe de sistema con una mejor presentación, y vista resumida de los resultados del benchmark, la cual muestro en la *Ilustración 3*.

## Results Overview


benchmark	
	benchmark-2
systemd-boot-total: Test: Total	183747
systemd-boot-total: Test: Kernel	442
systemd-boot-total: Test: Userspace	179008
PHORONIX-TEST-SUITE.COM	

Ilustración 3. Resumen de los resultados del benchmark

Como vemos en la *Ilustración 3*, solo se muestran los resultados de las opciones 1, 2 y 3 que se veían en la *Ilustración 1*. Esto es debido a que durante la prueba los otros 2 han fallado.

Los resultados mostrados son una media y están en milisegundos.

Por último, también podemos ver estos resultados más detalladamente en una serie de gráficas que se nos muestran en la página, las cuales podemos ver en la *Ilustración 4*.

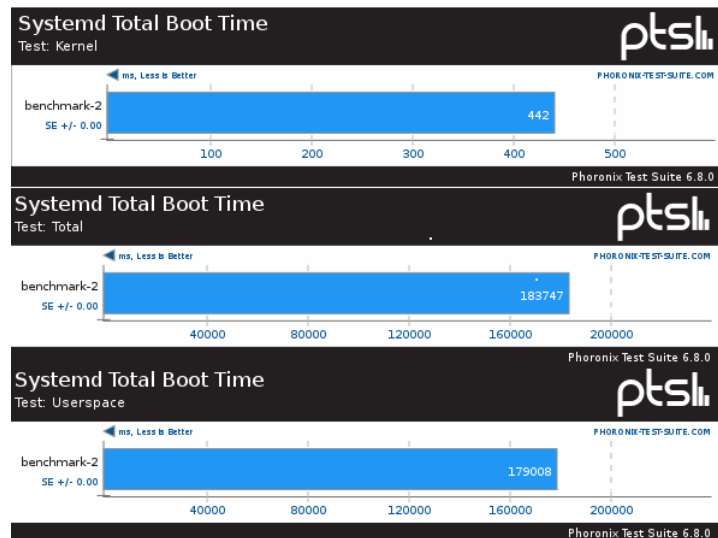


Ilustración 4. Gráficas con los resultados del benchmark

## 2ª Cuestión: De los parámetros que le podemos pasar al comando ¿Qué significa -c 5? ¿y -n 100? Monitoree la ejecución de ab contra alguna máquina (cualquiera) ¿cuántas “tareas” crea ab en el cliente?

La opción -c indica el número de peticiones múltiples que puede atender al mismo tiempo. Y la opción -n indica el número de peticiones que se van a usar para la prueba.

Voy a probar a ejecutar `ab -c 5 -n 100 http://localhost/` en mi servidor apache de CentOS, y obtengo el resultado que se muestra en la *Ilustración 5*, en la que como vemos ha atendido a las 100 peticiones, con una concurrencia de 5, sin problemas.

```
[AndMolLop 21/12/16 ~]# ab -c 5 -n 100 http://localhost/
This is ApacheBench, Version 2.3 <Revision: 1430300>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done


Server Software:      Apache/2.4.6
Server Hostname:      localhost
Server Port:          80

Document Path:        /
Document Length:      4897 bytes

Concurrency Level:     5
Time taken for tests:  0.324 seconds
Complete requests:     100
Failed requests:       0
Write errors:          0
Non-2xx responses:    100
Total transferred:     517900 bytes
HTML transferred:     489700 bytes
Requests per second:   308.68 [#/sec] (mean)
Time per request:      16.198 [ms] (mean)
Time per request:      3.240 [ms] (mean, across all concurrent requests)
Transfer rate:         1561.20 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0    1  0.3      1      3
Processing:  3    6 18.7      4     191
Waiting:    3    6 18.7      3     190
Total:      4    7 18.9      4     193

Percentage of the requests served within a certain time (ms)
 50%    4
 66%    5
 75%    5
 80%    5
 90%    6
 95%    9
 98%   13
 99%   193
100%   193 (longest request)
```

Ilustración 5. Ejecución de ab en CentOS

**3ª Cuestión: Ejecute ab contra las tres máquinas virtuales (desde el SO anfitrión a las máquinas virtuales de la red local) una a una (arrancadas por separado). ¿Cuál es la que proporciona mejores resultados? Muestre y coméntelos. (Use como máquina de referencia Ubuntu Server para la comparativa).**

Como la máquina anfitriona corre Windows 10, para poder ejecutar ab contra las máquinas virtuales lo más cómodo es instalar XAMPP primero en ella, para así tener el Apache Benchmark, el cual se localizará en la carpeta donde hayamos instalado XAMPP, .../xampp/apache/bin/ab.exe.

Una vez tengo XAMPP listo en la máquina anfitriona, arranco CentOS 7 y ejecuto ab contra él desde la máquina anfitriona por medio de la PowerShell, con la orden ./ab.exe -c 5 -n 100 http://ip\_CentOS/ como muestro en la *Ilustración 6*.

```
PS C:\xampp\apache\bin> ./ab.exe -c 5 -n 100 http://192.168.0.155/
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.0.155 (be patient).....done

Server Software:      Apache/2.4.6
Server Hostname:      192.168.0.155
Server Port:          80

Document Path:        /
Document Length:      4897 bytes

Concurrency Level:    5
Time taken for tests:  0.441 seconds
Complete requests:    100
Failed requests:       0
Non-2xx responses:    100
Total transferred:    517900 bytes
HTML transferred:     489700 bytes
Requests per second:  226.64 [#/sec] (mean)
Time per request:     22.061 [ms] (mean)
Time per request:     4.412 [ms] (mean, across all concurrent requests)
Transfer rate:        1146.28 [Kbytes/sec] received

Connection Times (ms)
      min      mean[+/-sd] median    max
Connect:    0       1   1.9      1      12
Processing:  4      20  21.0     13     159
Waiting:    3      16  13.9     12      83
Total:      6      21  20.9     14     159

Percentage of the requests served within a certain time (ms)
 50%    14
 66%    20
 75%    25
 80%    27
 90%    34
 95%    73
 98%    89
 99%   159
100%   159 (longest request)
PS C:\xampp\apache\bin> AndMolLop 22/12/16
```

*Ilustración 6. Ejecución de ab desde la máquina anfitriona contra CentOS*

Lo siguiente es ejecutarlo contra Ubuntu Server. Muestro el resultado en la *Ilustración 7*.

```
PS C:\xampp\apache\bin> ./ab.exe -c 5 -n 100 http://192.168.0.156/
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.0.156 (be patient).....done

Server Software:      Apache/2.4.18
Server Hostname:      192.168.0.156
Server Port:          80

Document Path:        /
Document Length:      11321 bytes

Concurrency Level:    5
Time taken for tests:  0.382 seconds
Complete requests:    100
Failed requests:       0
Total transferred:    1159500 bytes
HTML transferred:     1132100 bytes
Requests per second:  261.46 [#/sec] (mean)
Time per request:     19.124 [ms] (mean)
Time per request:     3.825 [ms] (mean, across all concurrent requests)
Transfer rate:        2960.53 [Kbytes/sec] received

Connection Times (ms)
      min     mean[+/-sd] median   max
Connect:    0       1   1.5      1    14
Processing:  2      17  20.5     10   105
Waiting:    1      13  17.2      7   101
Total:      3      19  20.4     12   106

Percentage of the requests served within a certain time (ms)
 50%    12
 66%    18
 75%    21
 80%    24
 90%    41
 95%    78
 98%    93
 99%   106
100%   106 (longest request)
PS C:\xampp\apache\bin> AndMoiLop 22/12/16
```

*Ilustración 7. Ejecución de ab desde la máquina anfitriona contra Ubuntu*

Por último, lo ejecutamos contra Windows server, aunque este funcione con IIS, obteniendo los resultados mostrados en la *Ilustración 8*.



```

PS C:\xampp\apache\bin> ./ab.exe -c 5 -n 100 http://192.168.0.157/
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.0.157 (be patient).....done

Server Software:      Microsoft-IIS/7.5
Server Hostname:      192.168.0.157
Server Port:          80

Document Path:        /
Document Length:      689 bytes

Concurrency Level:    5
Time taken for tests:  0.953 seconds
Complete requests:    100
Failed requests:       0
Total transferred:    93000 bytes
HTML transferred:     68900 bytes
Requests per second:  104.91 [#/sec] (mean)
Time per request:     47.660 [ms] (mean)
Time per request:     9.532 [ms] (mean, across all concurrent requests)
Transfer rate:        95.28 [kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0      1   2.4      1    19
Processing:  6     46 143.7      9   670
Waiting:    6     45 143.7      9   669
Total:      7     47 143.6     10   670

Percentage of the requests served within a certain time (ms)
 50%    10
 66%    16
 75%    20
 80%    22
 90%    40
 95%   668
 98%   670
 99%   670
100%   670 (longest request)
PS C:\xampp\apache\bin> AndMolLop 22/12/16

```

*Ilustración 8. Ejecución de ab desde la máquina anfitriona contra Windows Server*

Como vemos en todas las pruebas se le han mandado al servidor 100 peticiones, usando 5 clientes de manera concurrente. Y si nos fijamos en los resultados, vemos que Ubuntu Server es el más rápido en atender todas las peticiones, tardando 0.382 segundos, el siguiente es CentOS con un tiempo de 0.441 segundos y el último es Windows con 0.953 segundos.

Por lo que vemos que CentOS es un 13% más lenta que Ubuntu Server, y Windows Server es un 60% más lenta que Ubuntu Server. Por lo tanto, podemos ver que Ubuntu Server es el que mejor rendimiento nos da.

**4º Cuestión: Instale y siga el tutorial en <http://jmeter.apache.org/usermanual/build-web-test-plan.html> realizando capturas de pantalla y comentándolas. En vez de usar la web de jmeter, haga el experimento usando sus máquinas virtuales ¿coincide con los resultados de ab?**

Lo primero que vamos a hacer es descargarnos Jmeter y ejecutarlo. Una vez está corriendo le damos click derecho encima de Plan de Pruebas en el lateral izquierdo. Con esto se nos cambiará el cuadro derecho, mostrándonos el que vemos en la *Ilustración 9*, al cual le he cambiado el nombre, el número de hilos y el número de veces que se va a ejecutar.

*Ilustración 9. Configuración del Grupo de Hilos de JMeter*

Lo siguiente es especificar la dirección IP a la que vamos a hacerle el test, para ello vamos a hacer click derecho sobre el grupo de hilos definido en el panel izquierdo, y le damos a añadir > elemento de configuración > valores por defecto para peticiones HTTP. Con lo que el panel derecho nos cambiará al que muestro en la *Ilustración 10*. En el que he especificado la IP de mi servidor CentOS para empezar.

*Ilustración 10. Indicación de la IP del servidor al que hacer el test*

Ahora tenemos que crear la petición HTTP, para ello le damos click derecho sobre el grupo de hilos definido, y el damos a añadir > muestreador > petición HTTP. Y simplemente le ponemos un nombre y le definimos la ruta en la cual ponemos /. Esto lo muestro en la *Ilustración 11*.

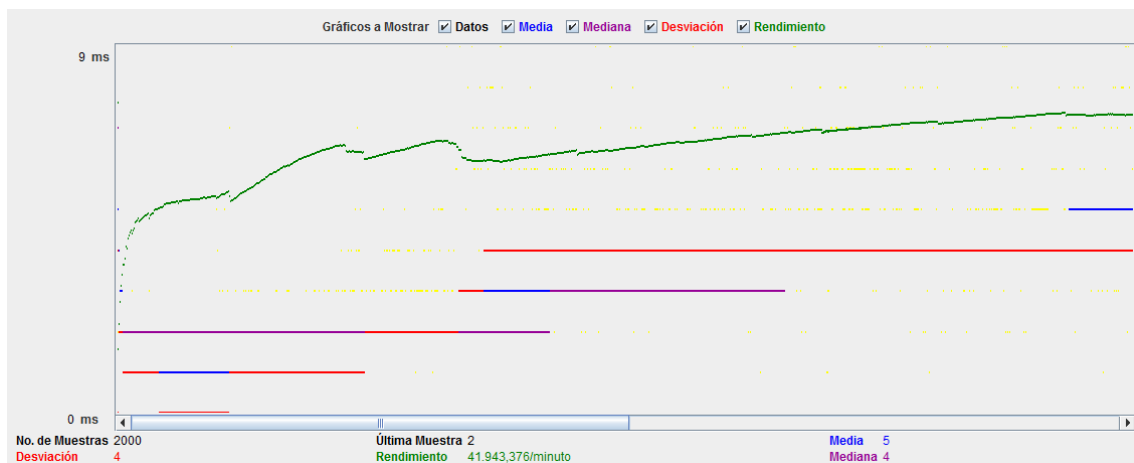
*Ilustración 11. Petición HTTP*

Por último, añadimos un receptor del tipo gráfico de resultados para que almacene los datos del test. Para ello le damos click derecho sobre el grupo de hilos en

el panel izquierdo y le damos a añadir > receptor > gráfico de resultados. En la *Ilustración 12* muestro esta última pantalla, en la cual solo hay que ponerle un fichero de salida.

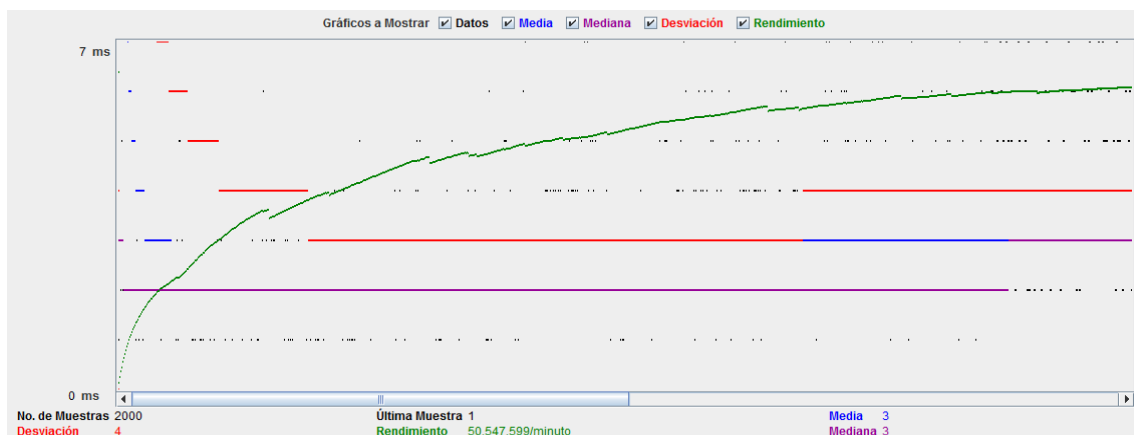
*Ilustración 12. Gráfico de resultados*

Con esto tenemos el test listo para lanzarlo dándole al botón de Play verde en la cabecera de JMeter, esto nos rellenará la gráfica de resultados. En mi caso para que la muestra sea significativa y se vean bien los resultados, he aumentado el número de bucles por cliente a 400. Y para CentOS obtengo los resultados que muestro en la *Ilustración 13*.



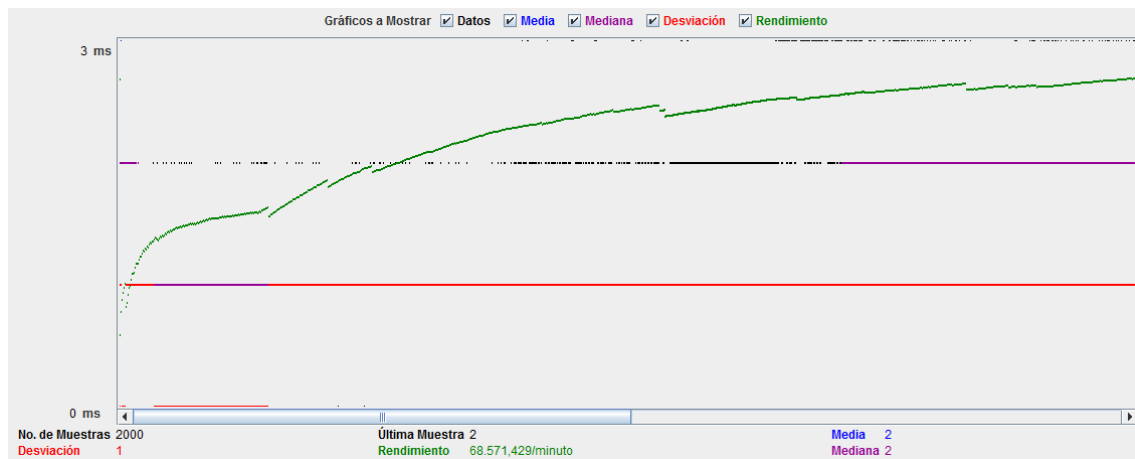
*Ilustración 13. Resultado del test en CentOS*

Ahora procedo a hacerle el test a Ubuntu Server, para el cual puedo aprovechar el mismo test, y solo hay que cambiar la IP del servidor en el Valor por Defecto para Peticiones HTTP. Tras cambiar esto y ejecutar el test, obtengo los resultados que se ven en la *Ilustración 14*.



*Ilustración 14. Resultado del test en Ubuntu Server*

Por último, lo ejecuto contra Windows Server, obteniendo los resultados mostrados en la *Ilustración 15*.



*Ilustración 15. Resultado del test en Windows Server*

Como vemos, los resultados en comparación a los que nos dio Apache Benchmark en el ejercicio anterior han cambiado y según nos muestra Windows Server es el que mejor rendimiento nos da por minuto pudiendo atender 68571429 peticiones por minuto con 5 clientes antes de saturarse, luego sería Ubuntu Server pudiendo atender 50547599 peticiones por minuto con el mismo número de clientes, y por último CentOS pudiendo atender 41943376 peticiones por minuto también con 5 clientes.

## **5º Cuestión: Programe un benchmark usando el lenguaje que desee. El benchmark debe incluir:**

### **1) Objetivo del benchmark.**

El objetivo del benchmark va a ser ver que lenguaje de programación ejecuta más rápidamente el algoritmo de ordenación por burbuja para vectores de distintos tamaños. Los lenguajes de programación elegidos han sido C++, Java, Perl, Python, Ruby y PHP.

Realmente podríamos comparar cualquier algoritmo, solo habría que cambiar la zona del código en la que yo he puesto el algoritmo de ordenación burbuja por cualquier otro.

Como dato decir que el vector siempre empieza ordenado de mayor a menor independientemente del número de elementos, esto es para que siempre sea el peor caso posible a encontrarnos para ordenar.

### **2) Métricas (unidades, variables, puntuaciones, etc.).**

Como lo que queremos comprobar es en que lenguaje se ejecuta más deprisa, la única variable que vamos a usar es el tiempo, el cual lo vamos a medir en milisegundos.

### **3) Instrucciones para su uso.**

Ejecutarlo es tan simple como ejecutar los distintos códigos fuentes, y se nos mostrará por pantalla el tiempo que ha tardado en ejecutarse el algoritmo de ordenación burbuja para ordenar un vector con un número de elementos definido en la primera variable del código llamada TAMANIO. De modo que si cambiamos esta variable podremos cambiar el tamaño del vector al que queramos.

#### 4) Ejemplo de uso analizando los resultados.

Paso ahora a ejecutar los diferentes códigos fuentes que nos darán los resultados. Para hacer una comparativa más detallada primero voy a ejecutarlo con un vector de un tamaño pequeño (1000 elementos), luego con uno de un tamaño más mediano (20000 elementos) y, por último, con un vector grande (50000 elementos).

Tras la ejecución de los códigos con estos distintos tamaños, podemos elaborar la siguiente gráfica a partir de los resultados obtenidos.



Como podemos apreciar, según aumenta el tamaño del vector con los datos, se van diferenciando mejor los tiempos de ejecución en cada

lenguaje. Así vemos que entre C++ y Java en la gráfica hay poca diferencia, pero si miramos la tabla de datos, veremos que Java es el doble de rápido.

Por lo tanto, se puede apreciar claramente, que Java es el mejor lenguaje para hacer este tipo de operaciones, seguido muy de cerca por C++, tanto, que incluso para una pequeña cantidad de datos C++ es mejor, como vemos en la tabla.

Por lo demás, podemos ver que el resto de lenguajes para realizar este tipo de operaciones no son muy buenos, siendo PHP el que peor respuesta da para grandes volúmenes de datos.

Los datos de la tabla se obtienen de la ejecución de estos programas, que dan los siguientes resultados mostrados en las *Ilustración 16* para 1000 datos, en la *Ilustración 17* para 20000 datos y en la *Ilustración 18* para 50000 datos.

```
AndMolLop 23/12/16 $ ./burbujac
8.57090950012207 milisegundos
AndMolLop 23/12/16 $ java burbujaj
49 milisegundos
AndMolLop 23/12/16 $ perl burbujap.pl
620.424 milisegundos
AndMolLop 23/12/16 $ python burbujapy.py
643.328905106 milisegundos
AndMolLop 23/12/16 $ ruby burbujar.rb
364.18964900000003 milisegundos
AndMolLop 23/12/16 $ php5 burbujaphp.php
512.01295852661 milisegundos
AndMolLop 23/12/16 $ █
```

*Ilustración 16. Resultados para 1000 datos*

```
AndMolLop 23/12/16 $ ./burbujac
2517.664909362793 milisegundos
AndMolLop 23/12/16 $ java burbujaj
1096 milisegundos
AndMolLop 23/12/16 $ perl burbujap.pl
86703.884 milisegundos
AndMolLop 23/12/16 $ python burbujapy.py
89739.2339706 milisegundos
AndMolLop 23/12/16 $ ruby burbujar.rb
45341.592842 milisegundos
AndMolLop 23/12/16 $ php5 burbujaphp.php
71099.757909775 milisegundos
AndMolLop 23/12/16 $ █
```

*Ilustración 17. Resultados para 20000 datos*

```

AndMolLop 23/12/16 $ ./burbujac
11038.51795196533 milisegundos
AndMolLop 23/12/16 $ java burbujaj
5153 milisegundos
AndMolLop 23/12/16 $ perl burbujap.pl
592593.987 milisegundos
AndMolLop 23/12/16 $ python burbujapy.py
671148.842812 milisegundos
AndMolLop 23/12/16 $ ruby burbujar.rb
434798.009967 milisegundos
AndMolLop 23/12/16 $ php5 burbujaphp.php
811681.19502068 milisegundos
AndMolLop 23/12/16 $ █

```

*Ilustración 18. Resultados para 50000 datos*

Para finalizar nuestro los distintos códigos fuentes usados:

- **C++**

```

/*****/

/*
    Medición de tiempos para el algoritmo de
    ordenación burbuja

    En c++

*/

/*****/

#include <time.h>

#include <stdio.h>

#include <sys/time.h>

double timeval_diff(struct timeval *a, struct timeval *b){
    return ((double)(a->tv_sec + (double) a-> tv_usec/1000000) - (double)(b->tv_sec + (double) b->
tv_usec/1000000));
}

int main(){

    //Primero generamos el peor de los casos para 20000 elementos,
    // es decir, un vector de 20000...1 ordenado de mayor a menor

    int TAMANIO = 50000;

    int numeros[TAMANIO];

```

```

int val = TAMANIO;

for(int i=0; i<TAMANIO; i++){
    numeros[i] = val;
    val--;
}

//Una vez creado este vector, pasamos a ordenarlo mediante burbuja
struct timeval t_ini, t_fin;

double secs;

int tmp;

gettimeofday(&t_ini, NULL);

for(int i=0; i<TAMANIO; i++){
    for(int k=i; k<TAMANIO; k++){
        if(numeros[k] < numeros[i]){
            tmp = numeros[k];
            numeros[k] = numeros[i];
            numeros[i] = tmp;
        }
    }
}

gettimeofday(&t_fin, NULL);

secs = timeval_diff(&t_fin, &t_ini);

printf("%.16g milisegundos\n", secs * 1000.0);

return 0;
}

```

- **Java**

```

/*****/

/*

```

Medición de tiempos para el algoritmo de



ordenación burbuja

En java

```
*/  
/*****/  
  
public class burbujaj{  
    public static void main(String arg[]){  
        int TAMANIO = 50000;  
        int numeros[] = new int[TAMANIO];  
        int val = TAMANIO;  
  
        for(int i=0; i<TAMANIO; i++){  
            numeros[i] = val;  
            val--;  
        }  
  
        long t_ini, t_fin;  
        int tmp;  
        t_ini = System.currentTimeMillis();  
        for(int i=0; i<TAMANIO; i++){  
            for(int j=i; j<TAMANIO; j++){  
                if(numeros[j] < numeros[i]){  
                    tmp = numeros[j];  
                    numeros[j] = numeros[i];  
                    numeros[i] = tmp;  
                }  
            }  
        }  
        t_fin = System.currentTimeMillis();  
        System.out.println(t_fin - t_ini + " milisegundos");  
    }  
}
```

- **Perl**

```
#
#
#     Medición de tiempos para el algoritmo de
#     ordenación burbuja
#     En perl
#
#

use Time::HiRes qw( gettimeofday );

$TAMANIO = 50000;

my @numeros;
my $val = $TAMANIO;

for(my $i=0;$i<$TAMANIO;$i++){
    $numeros[$i] = $val;
    $val--;
}

($seconds0, $microseconds0) = gettimeofday;
for(my $i=0;$i<@numeros;$i++){
    for(my $j=$i;$j<@numeros;$j++){
        if($numeros[$j] < $numeros[$i]){
            ($numeros[$i], $numeros[$j]) = ($numeros[$j], $numeros[$i]);
        }
    }
}

($seconds1, $microseconds1) = gettimeofday;

$seconds = ($seconds1 - $seconds0) * 1000;
$microseconds = ($microseconds1 - $microseconds0)/1000;
```

```
print $seconds + $microseconds . " milisegundos\n";
```

- **PHP**

```
<?php
```

```
/*
```

```
    Medicion de tiempos para el algoritmo de
```

```
    ordenacion burbuja
```

```
    En PHP
```

```
*/
```

```
$TAMANIO = 50000;
```

```
$numeros = array();
```

```
$val = $TAMANIO;
```

```
for($i=0; $i < $TAMANIO; $i++){
```

```
    $numeros[] = $val;
```

```
    $val--;
```

```
}
```

```
$t_ini = microtime(true);
```

```
for($i=0; $i<sizeof($numeros); $i++){
```

```
    for($j=$i; $j<sizeof($numeros); $j++){
```

```
        if($numeros[$j] < $numeros[$i]){
```

```
            $tmp = $numeros[$j];
```

```
            $numeros[$j] = $numeros[$i];
```

```
            $numeros[$i] = $tmp;
```

```
        }
```

```
    }
```

```
}
```

```
$t_fin = microtime(true);  
$t_total = ($t_fin - $t_ini) * 1000;
```

```
echo $t_total . " milisegundos";
```

```
?>
```

- **Python**

```
#
```

```
#
```

```
#     Medicion de tiempos para el algoritmo de
```

```
#     ordenacion burbuja
```

```
#     En python
```

```
#
```

```
#
```

```
from time import time
```

```
tamano = 50000
```

```
numeros = []
```

```
val = tamano
```

```
for i in range(0, tamano):
```

```
    numeros.append(val)
```

```
    val = val-1
```

```
t_ini = time()
```

```
i=0
```

```
while(i < len(numeros)):
```

```
    k=i
```

```
    while(k < len(numeros)):
```

```
        if(numeros[k] < numeros[i]):
```

```

        tmp = numeros[k]
        numeros[k] = numeros[i]
        numeros[i] = tmp

        k=k+1

    i=i+1

t_fin = time()
t_total = (t_fin - t_ini) * 1000

print t_total , " milisegundos"

```

- **Ruby**

```

#
#
#   Medicion de tiempos para el algoritmo de
#   ordenacion burbuja
#   En ruby
#
#

tamanio = 50000

numeros = []
val = tamanio

for i in (0..tamanio)
    numeros << val
    val-=1
end

t_ini = Time.now
i=0
while i < numeros.length do

```

```
j = i
while j < numeros.length do
  if(numeros[j] < numeros[i])
    tmp = numeros[j]
    numeros[j] = numeros[i]
    numeros[i] = tmp
  end
  j += 1
end
i+=1
end

t_fin = Time.now
t_total = (t_fin - t_ini) * 1000

puts t_total.to_s + ' milisegundos'
```

## **Bibliografía:**

- Cuestión 1:
  - [1] Benchmarks disponibles en Phoronix Test Suite con sus descripciones <https://openbenchmarking.org/tests/pts&s=c>