

Funcionalidades y comprobación de la orden MPSTAT de forma experimental

Ingeniería de Servidores

Universidad de Granada

Resumen

La carga de trabajo de la CPU son varios procesos que usan una porción de tiempo de la CPU o que están en la cola de procesos preparados esperando a entrar en la CPU. En este texto explicamos cómo afecta la carga de trabajo al sistema con la orden *mpstat* centrándonos en los campos *%idle*, *%usr* e *%iowait*. Esto lo haremos con un script, el cual nos ayudará a meter carga de trabajo y por lo tanto podremos ver los cambios que se producen en nuestro sistema con la orden *mpstat*. Guardaremos dichos resultados del script redirigiendo la salida a un *resultados.txt* para posteriormente analizar estos datos y ver los resultados de forma gráfica.

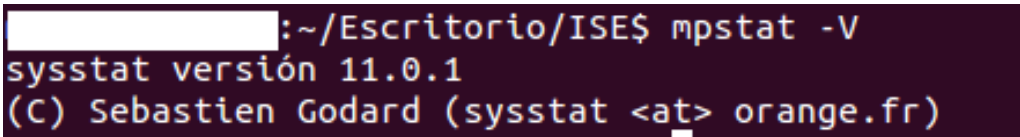
1 Introducción

Antes de nada, debemos saber qué mide y cuál es la función que realiza la orden *mpstat*. *Mpstat* nos muestra las actividades del o los procesadores (en caso de que tengamos varios núcleos). También nos proporciona un promedio de las actividades de todos los núcleos en conjunto. Recordar que para usar esta orden es necesario tener instalado el paquete *sysstat*, el cual podremos instalarlo con la orden *sudo apt-get install sysstat*

1.1 Opciones de mpstat

La orden *mpstat* nos proporciona una serie de opciones:

- -V. Nos muestra la versión que tenemos instalada de *mpstat* [1]



```

[redacted]:~/Escritorio/ISE$ mpstat -V
sysstat versión 11.0.1
(C) Sebastien Godard (sysstat <at> orange.fr)
```

Figura 1.1.1: *mpstat -V*

- -P. Nos da la opción de elegir sobre qué procesador queremos que se muestren las estadísticas. Si lo ejecutamos con la opción *ALL*, se nos mostrarán las estadísticas de todos los procesadores disponibles en la máquina.
- -A. Es equivalente a poner *mpstat -u, -l ALL* ó *-P ALL*

La sintaxis de la orden *mpstat* acepta los siguientes parámetros:

mpstat [opciones] [<intervalo> [número de muestras]], siendo <opciones> las explicadas anteriormente, <intervalo> los segundos con los que recoge una muestra y el <número de muestras> , como su propio nombre indica, el número de veces que nos imprime los resultados por pantalla.

2 Campos de mpstat

- %usr: Porcentaje de uso en el nivel de usuario
- %sys: Porcentaje de uso en el nivel del sistema (kernel). En el cómputo no se incluyen interrupciones
- %iowait: Porcentaje del tiempo esperando operaciones de entrada / salida. Principalmente discos, red,...
- %nice: Porcentaje del tiempo que un proceso de usuario se ejecuta con la prioridad variada con nice
- %irq: Porcentaje del tiempo sirviendo interrupciones de hardware
- %soft: Porcentaje del tiempo sirviendo interrupciones de software. Normalmente se tratan de llamadas al sistema, que se implementan mediante interrupciones software
- %steal: En kernels superiores al 2.6.11 son los ciclos dedicados a tareas como la virtualización [1]
- %idle: Tiempo en la que la CPU ha estado sin trabajo y sin estar esperando entradas / salidas

2.1 Fiabilidad de la orden mpstat

Como se nos dice en [2] *“Es muy probable que nos fijemos sólo en el indicador de %user para decidir cómo de ocupada está la CPU. Aunque en algunos tipos de carga puede ser suficiente, no se puede generalizar porque un %user al 10% no significa para nada que la CPU esté libre, sino que puede estar esperando discos. Por lo tanto, tenemos que fijarnos también en el parámetro %iowait.*

Una buena forma de ver rápidamente el consumo de CPU es fijándonos en el %idle, pero por supuesto debemos fijarnos también en el resto de indicadores (los explicados anteriormente).

Pero lo que realmente debemos preguntarnos es: ¿es fiable esta información? La respuesta es sí y no, como todo muestreo tiene sus errores. En este caso la CPU se recoge mediante la interrupción timer, que mira en ese momento qué está haciendo la CPU. Por lo tanto podremos imaginar la siguiente situación:

- *Salta el timer, la CPU está en el handle idle, por lo que se registra como tal y sigue*
- *Ocurre una interrupción y se sirve antes que vuelva a saltar el timer*
- *Salta el timer, la CPU vuelve a estar con el handler idle, por lo que consideramos que todo el período ha estado idle*

Por tanto, no significa que los datos no sean fiables, significa que, como en todo muestreo, hay un porcentaje de error a tener en cuenta.”

3 mpstat de forma experimental

En relación con la parte práctica, hemos realizado un programa que multiplica matrices, en lenguaje C++, con la finalidad de aumentar la carga de trabajo del sistema, ya que la multiplicación de matrices con bastantes filas y columnas es bastante pesado para el sistema en general. Tomamos una serie de muestras aumentando en 200 cada vez el tamaño de las

matrices para ver con la orden `mpstat -P ALL 1 85`, siendo 1 el intervalo y 85 el número de muestras (ya se lo hemos explicado en el apartado 1.1) cómo nuestra carga aumenta considerablemente desde el inicio hasta el fin de la ejecución del programa. Realizaremos una serie de gráficas con dichos datos.

3.1 Hardware

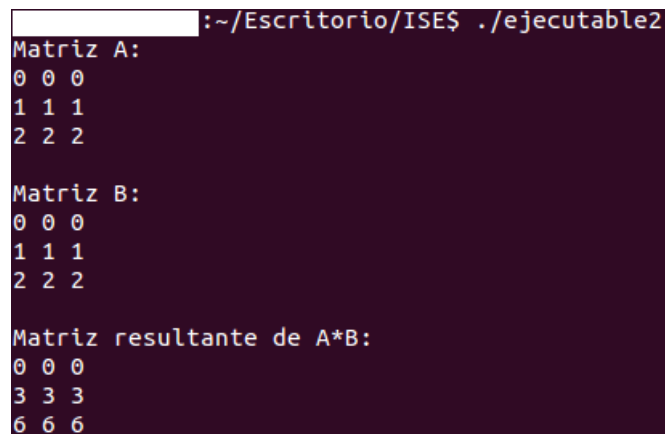
Realizaremos las pruebas sobre una máquina virtual montada en VMWare con 4 GB de RAM y un procesador Intel i7-4720HQ, el cual dispone de 8 núcleos: 4 reales y 4 virtuales. Para verlo con más claridad estas pruebas configuraremos nuestra máquina virtual con un único core. Para ver más en detalle la información sobre el procesador de esta máquina introduciremos el comando `cat /proc/cpuinfo` y redirigiremos la salida a un archivo de texto plano llamado `información.txt` quedando el comando de la siguiente forma `'cat /proc/cpuinfo > información.txt'`. El archivo que nos genera es el siguiente:

```
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 60
model name    : Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
stepping      : 3
microcode     : 0x1e
cpu MHz       : 2594.008
cache size    : 6144 KB
physical id   : 0
siblings      : 1
core id       : 0
cpu cores     : 1
apicid        : 0
initial apicid : 0
fpu           : yes
fpu_exception : yes
cpuid level   : 13
wp            : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts mmx
fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology tsc_reliable
nonstop_tsc aperfmperf eagerfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt
tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm ida arat epb pln pts dtherm fsgsbase tsc_adjust
bmi1 avx2 smep bmi2 invpcid xsaveopt
bugs          :
bogomips      : 5188.01
clflush size  : 64
cache_alignment : 64
address sizes  : 42 bits physical, 48 bits virtual
power management:
```

Como podemos observar en este archivo del sistema, se puede apreciar que tan sólo tenemos un core, ya que si no nos habría generado la misma información x4 (4 cores) especificando en la primera línea de cada uno el core al que se refiere, (processor : n), siendo n el número de core.

3.2 Programa de multiplicación de matrices

En primer lugar vamos a realizar una prueba sencilla con dos matrices de orden 3x3 [3] (Figura 3.2.1) para ver si realmente nuestro programa está funcionando de forma correcta. En este programa tendremos una variable global número para poder aumentar fácilmente el tamaño de las 2 matrices. Este programa tan solo trabajará con matrices cuadradas. Los valores de las matrices se introducen de forma automática mediante dos for anidados desde 0 hasta número introduciendo en cada posición de la matriz el valor de i o de j.



```
~/Escritorio/ISE$ ./ejecutable2
Matriz A:
0 0 0
1 1 1
2 2 2

Matriz B:
0 0 0
1 1 1
2 2 2

Matriz resultante de A*B:
0 0 0
3 3 3
6 6 6
```

Figura 3.2.1: Multiplicación de matrices

El código es el siguiente:

```
#include <iostream>

#include <fstream>

using namespace std;

int numero = 5000;

void multmat(int **A, int **B, int **C);

int main() {

    int i, j, k;

    int **A, **B, **C;

    A = new int* [numero], B = new int* [numero], C = new int* [numero];

    for(j=0; j<numero; j++){

        A[j] = new int [numero], B[j] = new int [numero], C[j] = new int [numero];

    }

    for(i=0; i<numero; i++){

        for(j=0; j<numero; j++){

            A[i][j] = i;
```

```

        }
    }

    for(i=0; i<numero; i++){

        for(j=0; j<numero; j++){

            B[i][j] = i;

        }

    }

    multmat(A, B, C);

    for(i=0; i<numero; i++){

        for(j=0; j<numero; j++){

            cout << A[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

    for(i=0; i<numero; i++){

        for(j=0; j<numero; j++){

            cout << B[i][j] << " ";

        }

        cout << endl;

    }

    cout << endl;

    for(i=0; i<numero; i++){

        for(j=0; j<numero; j++){

            cout << C[i][j] << " ";

        }

        cout << endl;

    }

    delete [] A, B, C; //Desasigna la memoria

    return 0;

}

void multmat(int **A, int **B, int **C){

    int i, j, k;

```

```

for(i=0; i<numero; i++){

    for(j=0; j<numero; j++){

        C[i][j]=0;

        for(k=0; k<numero; k++){

            C[i][j]= C[i][j]+A[i][k]*B[k][j];

        }

    }

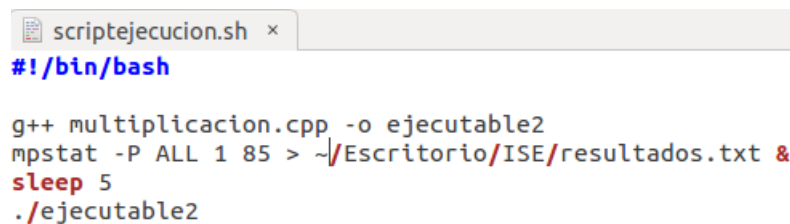
}

}

```

3.3 Ejecución

Para facilitar la ejecución vamos a realizar un script [4] (Figura 3.3.1). En la primera línea compilaremos el programa para únicamente tener que modificar la variable global número y poder seguir recogiendo muestras. En la segunda línea comenzaremos la monitorización del sistema con mpstat y los parámetros especificados anteriormente. Finalmente rediregiremos la salida de este script con > resultados.txt y se nos irá almacenando la información del sistema con el tamaño de matriz que hubiésemos especificado. En la tercera línea hemos puesto un sleep(5) para que no se ejecute la multiplicación de matrices hasta pasados 5 segundos. Esto lo hemos hecho para que se vea cómo en los primeros 5 segundos no hay carga de trabajo y a los 5 segundos empieza a subir el %usr.



```

scriptejecucion.sh x
#!/bin/bash

g++ multiplicacion.cpp -o ejecutable2
mpstat -P ALL 1 85 > ~/Escritorio/ISE/resultados.txt &
sleep 5
./ejecutable2

```

Figura 3.3.1: Script

Tras ejecutar este script nos ha generado el resultados.txt donde lo que nos interesa es la media, el cuál es el valor que hemos cogido para las gráficas

Media:	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
Media:	all	4,81	0,00	0,27	0,21	0,00	0,01	0,00	0,00	0,00	94,69
Media:	0	4,81	0,00	0,27	0,21	0,00	0,01	0,00	0,00	0,00	94,69

Figura 3.3.2: Resultado de la media

3.3.1 Resultados de forma gráfica

Para asegurarnos de que la carga de trabajo justo antes de que tomemos las muestras es baja y no hay demasiados procesos en segundo plano en ejecución, ejecutaremos la orden htop [5], la cual nos muestra de forma gráfica el porcentaje de uso de la CPU, de memoria y de swap, además de la carga media del sistema y la tareas que están en ejecución. Lo veremos en la siguiente imagen (Figura 3.3.1.1). También decir que hemos matado algunos procesos que estaban corriendo en segundo plano en nuestra máquina, entre ellos tomcat, con el comando kill PID.

CPU[|||||] 0.5%

Mem[|||||] 556/3936MB

Swp[|] 0/4092MB

Tasks: 105, 192 thr; 1 running

Load average: 0.23 0.46 0.22

Uptime: 00:03:30

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2500		20	0	30128	3668	2920	R	0.5	0.1	0:01.31	htop
2201		20	0	1096M	98172	66280	S	0.5	2.4	0:03.10	compiz
1147	root	20	0	339M	71700	35064	S	0.5	1.8	0:03.24	/usr/bin/X -cor
1560	www-data	20	0	352M	6036	2480	S	0.5	0.1	0:00.09	/usr/sbin/apach
1547	www-data	20	0	352M	6036	2480	S	0.5	0.1	0:00.07	/usr/sbin/apach
2246		20	0	215M	25016	21760	S	0.0	0.6	0:00.47	/usr/lib/vmware
867	root	20	0	155M	10896	8424	S	0.0	0.3	0:00.40	/usr/sbin/vmtoo
2411		20	0	607M	32836	26000	S	0.0	0.8	0:00.60	/usr/lib/gnome-
1950		20	0	22404	200	0	S	0.0	0.0	0:00.03	upstart-dbus-br
979	root	20	0	297M	11016	7628	S	0.0	0.3	0:00.36	/usr/lib/accoun
1498	www-data	20	0	352M	6036	2480	S	0.0	0.1	0:00.08	/usr/sbin/apach
955	root	20	0	197M	7936	7144	S	0.0	0.2	0:00.16	./ManagementAge
1023	root	20	0	197M	7936	7144	S	0.0	0.2	0:00.07	./ManagementAge
1499	www-data	20	0	352M	6036	2480	S	0.0	0.1	0:00.10	/usr/sbin/apach
1494	root	20	0	71564	4388	3200	S	0.0	0.1	0:00.02	/usr/sbin/apach
1022	root	20	0	197M	7936	7144	S	0.0	0.2	0:00.03	./ManagementAge
1049	root	20	0	297M	11016	7628	S	0.0	0.3	0:00.09	/usr/lib/accoun

Figura 3.3.1.1: htop

Como podemos observar, el porcentaje de uso de la CPU oscila entre 0.5% y 1.4%. Por lo tanto, la carga media de trabajo (Load average) es baja.

Hecho esto, comenzamos a realizar nuestras muestras con la ejecución del script que explicamos anteriormente.

Una vez cogidas las medias del %usr, %idle e %iowait lo representaremos mediante dos gráficas, en las que el %iowait aparecerá en ambas ya que muestra si la carga del sistema se muestra afectada por operaciones de entrada / salida. Esto es importante saberlo para ver la fiabilidad de estos resultados que hemos obtenido.

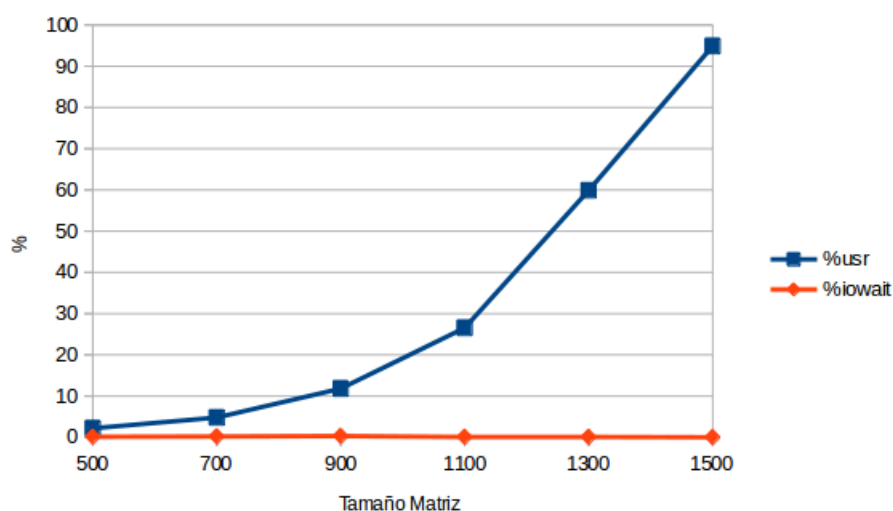


Figura 3.3.1.2: %usr e %iowait

Como se puede observar en la gráfica (Figura 3.3.1.2), el %iowait nunca supera el 1%. Esto nos indica que el procesador no está ejecutando en segundo plano tareas de entrada / salida, por lo que está dedicando todo su tiempo a la ejecución de nuestro programa. El %usr nos muestra el uso de la CPU, como es normal, éste sube conforme al tamaño de las matrices.

En esta gráfica (Figura 3.3.1.2) se ve claramente cómo a más tamaño de las matrices (eje x), más carga de trabajo para el sistema (eje y).

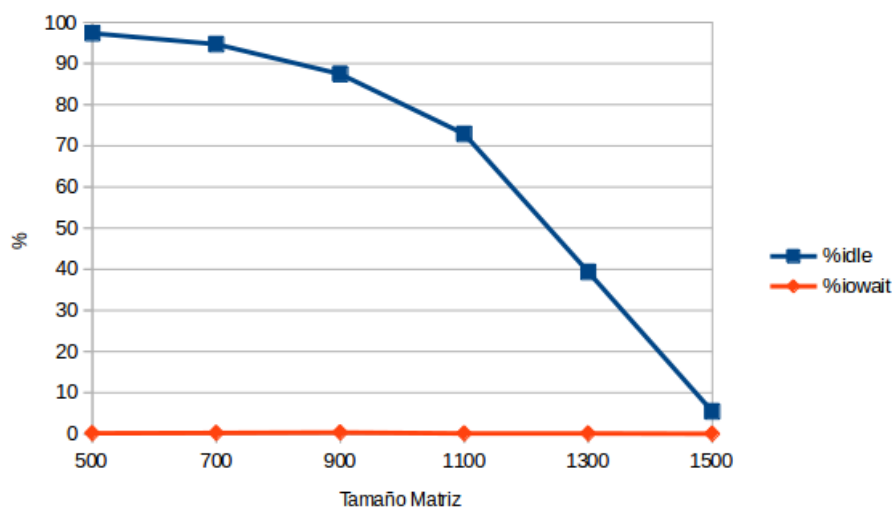


Figura 3.3.1.3: %idle e %iowait

Como podemos observar en esta figura (Figura 3.3.1.3) el %iowait no supera nunca el 1%. Por lo tanto, podremos decir que el procesador dedica todo su tiempo a la ejecución de nuestro programa. En cambio, el %idle nos muestra la no utilización de la CPU, como es normal, cuando el tamaño de las matrices son bajos, la no utilización será alta. Por consecuencia, a más tamaño de las matrices, menor será el tiempo de no utilización.

En esta gráfica (Figura 3.3.1.3) se ve claramente cómo a más tamaño de las matrices (eje x), menos porcentaje de no utilización de la CPU (eje y).

4 Conclusiones

En este estudio hemos podido ver cómo los campos %usr, %idle e %iowait son fiables tal y como observamos en las gráficas en el momento en el que introducíamos carga de trabajo en nuestro sistema. El comando mpstat responde de la forma esperada ante esta carga de trabajo.

En definitiva, con los datos que hemos ido recopilando a lo largo del estudio de la orden mpstat, podemos afirmar que dicha orden es fiable.

5 Referencias

- [1] http://linuxcommand.org/man_pages/mpstat1.html [consultado 20-Noviembre-2015]
- [2] <http://systemadmin.es/2009/11/como-leer-el-uso-de-cpu-en-el-top-o-el-mpstat> [consultado 23-Noviembre-2015]
- [3] Prácticas de la asignatura Arquitectura de Computadores que se imparte en la UGR
- [4] Prácticas de la asignatura Sistemas Operativos que se imparte en la UGR
- [5] <http://linux.die.net/man/1/htop> [consultado 25-Noviembre-2015]