

Sintonización, optimización y alta disponibilidad

Remo Suppi Boldrito

PID_00174430



Universitat Oberta
de Catalunya

www.uoc.edu

Índice

Introducción	5
Objetivos	6
1. Sintonización, optimización y alta disponibilidad	7
1.1. Aspectos básicos	7
1.1.1. Monitorización sobre UNIX System V	8
1.1.2. Optimización del sistema.....	13
1.1.3. Optimizaciones de carácter general.....	17
1.1.4. Configuraciones complementarias	18
1.1.5. Resumen de acciones para mejorar un sistema	21
1.2. Monitorización	24
1.2.1. Munin	24
1.2.2. Monit	25
1.2.3. MRTG	27
1.2.4. Otras herramientas	28
1.3. Alta disponibilidad en Linux (High-Availability Linux)	29
1.3.1. Guía breve de instalación de Heartbeat en Debian....	30
Actividades	33
Bibliografía	33

Introducción

Un aspecto fundamental, una vez que el sistema está instalado, es la configuración y adaptación del sistema a las necesidades del usuario y que las prestaciones del sistema sean lo más adecuadas posible a las necesidades que de él se demandan. GNU/Linux es un sistema operativo eficiente que permite un grado de configuración excelente y una optimización muy delicada de acuerdo a las necesidades del usuario. Es por ello que, una vez realizada una instalación (o en algunos casos una actualización), deben hacerse determinadas configuraciones vitales en el sistema. Si bien el sistema “funciona”, es necesario efectuar algunos cambios (adaptación al entorno o sintonización) para permitir que estén cubiertas todas las necesidades del usuario y de los servicios que presta la máquina. Esta sintonización dependerá de dónde se encuentre funcionando la máquina y en algunos casos se realizará para mejorar el rendimiento del sistema, mientras que en otros (además), por cuestiones de seguridad. Cuando el sistema está en funcionamiento, es necesario monitorizarlo para ver su comportamiento y actuar en consecuencia. Si bien es un aspecto fundamental, la sintonización de un sistema operativo muchas veces se relega a la opinión de expertos o gurús de la informática; pero conociendo los parámetros que afectan al rendimiento, es posible llegar a buenas soluciones haciendo un proceso cíclico de análisis, cambio de configuración, monitorización y ajustes. En este módulo se verán las principales herramientas para monitorizar un sistema GNU/Linux, como son Munin, Monit, MRTG, y se darán indicaciones de cómo sintonizar el sistema a partir de la información obtenida.

Otro aspecto importante en la actualidad de los servidores de sistemas de la información es la alta disponibilidad.

La alta disponibilidad (*high availability*) es un protocolo de diseño del sistema y su implementación asociada que asegura un cierto grado absoluto de continuidad operacional durante períodos largos de tiempo. El término *disponibilidad* se refiere a la habilidad de la comunidad de usuarios para acceder al sistema, enviar nuevos trabajos, actualizar o alterar trabajos existentes o recoger los resultados de trabajos previos. Si un usuario no puede acceder al sistema se dice que está *no disponible*.

De entre todas las herramientas que existen para tratar estos aspectos (Heartbeat, Ldirectord y LVS –Linux Virtual Server–, Piranha, UltraMonkey, Kimberlite, etc), en este módulo analizaremos algunas que permiten gestionar esta infraestructura redundante, como Heartbeat o Kimberlite.

La seguridad se estudia en el módulo “Administración de seguridad”.



Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

- 1.** Analizar y determinar las posibles pérdidas de prestaciones de un sistema.
- 2.** Solucionar problemas de sintonización del sistema.
- 3.** Instalar y analizar las diferentes herramientas y su integración para resolver los problemas de eficiencias.
- 4.** Analizar las herramientas que permiten tener un sistema en alta disponibilidad.

1. Sintonización, optimización y alta disponibilidad

1.1. Aspectos básicos

Antes de conocer cuáles son las técnicas de optimización, es necesario enumerar las causas que pueden afectar a las prestaciones de un sistema operativo [9]. Entre estas, se pueden mencionar:

1) **Cuellos de botella en los recursos:** la consecuencia es que todo el sistema irá más lento porque existen recursos que no pueden satisfacer la demanda a la que se les somete. El primer paso para optimizar el sistema es encontrar estos cuellos de botella y determinar por qué ocurren, conociendo sus limitaciones teóricas y prácticas.

2) **Ley de Amdahl:** según esta ley, “hay un límite de cuánto puede uno mejorar en velocidad una cosa si solo se optimiza una parte de ella”; es decir, si se tiene un programa que utiliza el 10% de la CPU y se optimiza reduciendo la utilización en un factor 2, el programa mejorará sus prestaciones (*speedup*) en un 5%, lo cual puede significar un tremendo esfuerzo no compensado por los resultados.

3) **Estimación del *speedup*:** es necesario estimar cuánto mejorará las prestaciones el sistema para evitar esfuerzos y costes innecesarios. Se puede utilizar la ley de Amdahl para valorar si es necesaria una inversión, en tiempo o económica, en el sistema.

4) **Efecto burbuja:** siempre se tiene la sensación de que cuando se encuentra la solución a un problema, surge otro. Una manifestación de este problema es que el sistema se mueve constantemente entre problemas de CPU y problemas de entrada/salida, y viceversa.

5) **Tiempo de repuesta frente a cantidad de trabajo:** si se cuenta con veinte usuarios, mejorar en la productividad significará que todos tendrán más trabajo hecho al mismo tiempo, pero no mejores respuestas individualmente; podría ser que el tiempo de respuesta para algunos fuera mejor que para otros. Mejorar el tiempo de respuesta significa optimizar el sistema para que las tareas individuales tarden lo menos posible.

6) **Psicología del usuario:** dos parámetros son fundamentales:

- a) el usuario generalmente estará insatisfecho cuando se produzcan variaciones en el tiempo de respuesta; y
- b) el usuario no detectará mejoras en el tiempo de ejecución menores del 20%.

7) **Efecto prueba:** las medidas de monitorización afectan a las propias medidas. Se debe ir con cuidado cuando se realizan las pruebas por los efectos colaterales de los propios programas de medida.

8) **Importancia de la media y la variación:** se deben tener en cuenta los resultados, ya que si se obtiene una media de utilización de CPU del 50% cuando ha sido utilizada 100, 0, 0, 100, se podría llegar a conclusiones erróneas. Es importante ver la variación sobre la media.

9) **Conocimientos básicos sobre el hardware del sistema a optimizar:** para mejorar una cosa es necesario “conocer” si es susceptible de mejora. El encargado de la optimización deberá conocer básicamente el hardware subyacente (CPU, memorias, buses, caché, entrada/salida, discos, vídeo, etc.) y su interconexión para poder determinar dónde están los problemas.

10) **Conocimientos básicos sobre el sistema operativo a optimizar:** del mismo modo que en el punto anterior, el usuario deberá conocer aspectos mínimos sobre el sistema operativo que pretende optimizar, entre los cuales se incluyen conceptos como procesos e hilos o *threads* (creación, ejecución, estados, prioridades, terminación), llamadas al sistema, *buffers* de caché, sistema de archivos, administración de memoria y memoria virtual (paginación, *swap*) y tablas del núcleo (*kernel*).

1.1.1. Monitorización sobre UNIX System V

El `/proc` lo veremos como un directorio, pero en realidad es un sistema de archivos ficticio, es decir, no existe sobre el disco y el núcleo lo crea en memoria. Se utiliza para proveer de información sobre el sistema (originalmente sobre procesos, de aquí el nombre), información que luego será utilizada por todos los comandos que veremos a continuación. Una vista de este directorio es:

```

1      ...  asound      ioports      sched_debug
124    ...  buddyinfo   irq          scsi
125    ...  bus         kallsyms     self
1258   ...  cgroups     kcore        slabinfo
126    ...  cmdline    key-users    stat
127    ...  cpuinfo     kmsg         swaps
1601   ...  crypto      kpagecount   sys
1612   ...  devices     kpageflags   sysrq-trigger
1851   ...  diskstats   loadavg      sysvipc
1886   ...  dma         locks        timer_list
1930   ...  driver      meminfo      timer_stats
1941   ...  execdomains misc          tty
1951   ...  fb          modules      uptime
1963   ...  filesystems mounts        version
1964   ...  fs          mtrr         vmallocinfo
2      ...  ide         net           vmstat
2012   ...  interrupts  pagetypeinfo zoneinfo
2049   ...  acpi        iomem        partitions

```

Hay un conjunto de archivos y directorios. A continuación veremos algunos de los más interesantes*:


*Consúltese la página del manual para obtener más información.


```

/proc/1: un directorio con la información del proceso 1 (el número del directorio es el
PID del proceso).
/proc/cpuinfo: información sobre la CPU (tipo, marca, modelo, prestaciones, etc.).
/proc/devices: lista de dispositivos configurados en el núcleo.
/proc/dma: canales de DMA utilizados en ese momento.
/proc/filesystems: sistemas de archivos configurados en el núcleo.
/proc/interrupts: muestra qué interrupciones están en uso y cuántas de ellas se han
procesado.
/proc/ioports: ídem con los puertos.
/proc/kcore: imagen de la memoria física del sistema.
/proc/kmsg: mensajes generados por el núcleo, que luego son enviados a syslog.
/proc/ksyms: tabla de símbolos del núcleo.
/proc/loadavg: carga del sistema.
/proc/meminfo: información sobre la utilización de memoria.
/proc/modules: módulos cargados por el núcleo.
/proc/net: información sobre los protocolos de red.
/proc/stat: estadísticas sobre el sistema.
/proc/uptime: desde cuándo el sistema está funcionando.
/proc/version: versión del núcleo.

```

Estos archivos se construyen de forma dinámica cada vez que se visualiza el contenido y el núcleo del sistema operativo los provee en tiempo real. Es por ello que se denomina *sistema de archivos virtual* y el contenido de los archivos y directorios va cambiando en forma dinámica con los datos actualizados. De este modo se puede considerar el `/proc/` como una interfaz entre el núcleo de Linux y el usuario y es una forma sin ambigüedades y homogénea de presentar información interna y puede ser utilizada para las diversas herramientas/comandos de información/sintonización/control que utilizaremos regularmente. Es interesante, por ejemplo, ver la salida de comando `mount` y el resultado de la ejecución `more /proc/mounts`: es totalmente equivalente!



En los siguientes subapartados se enseñará cómo obtener y modificar la información del núcleo de Linux trabajando con el sistema de archivos `/proc`.

Se debe tener en cuenta que estos archivos son visibles (texto), pero algunas veces los datos están “en crudo” y son necesarios comandos para interpretarlos, que serán los que veremos a continuación. Los sistemas compatibles UNIX SV utilizan los comandos `sar` y `sadc` para obtener estadísticas del sistema. En Debian es `atsar` (y `atsadc`), que es totalmente equivalente a los que hemos mencionado y posee un conjunto de parámetros que nos permiten obtener información de todos los contadores e información sin procesar del `/proc`. Debian también incluye el paquete `sysstat` que contiene los comandos `sar` (información general de la actividad del sistema), `iostat` (utilización CPU y de E/S), `mpstat` (informes globales por procesador), `pidstat` (estadísticas de procesos), `sadf` (muestra información del `sar` en varios formatos). El comando `atsar` lee contadores y estadísticas del fichero `/proc` y las muestra por la salida estándar. La primera forma de llamar al comando es (ejecutarlo como `root` o agregar el usuario a la categoría correspondiente del `sudoers` para ejecutar con el `sudo`):

```
atsar opciones t [n]n
```

Donde muestra la actividad en `n` veces cada `t` segundos con una cabecera que muestra los contadores de actividad (el valor por defecto de `n = 1`). La segunda forma de llamarlo es:

```
atsar -opciones -s time -e time -i sec -f file -n day#
```

El comando extrae datos del archivo especificado por `-f` (que por defecto es `/var/log/atsar/atsarxx`, siendo `xx` el día del mes) y que fueron previamente guardados por `atsadc` (se utiliza para recoger los datos, salvarlos y procesarlos y en Debian está en `/usr/lib/atsar`). El parámetro `-n` puede ser utilizado para indicar el día del mes y `-s, -e` la hora de inicio y final, respectivamente. Para activar `atsadc`, por ejemplo, se podría incluir en `/etc/cron.d/atsar` una línea como la siguiente:

```
@reboot root test -x /usr/lib/atsadc && /usr/lib/atsar/atsadc /var/log/atsar/atsa'date +%d'
10,20,30,40,50 * * * * root test -x /usr/lib/atsar/atsa1 && /usr/lib/atsar/atsa1
```

La primera línea crea el archivo después de un reinicio y la segunda guarda los datos cada 10 minutos con el *shell script* `atsa1`, que llama al `atsadc`. En `atsar` (o `sar`), las opciones se utilizan para indicar qué contadores hay que mostrar y algunos de ellos son:

Opciones	Descripción
u	Utilización de CPU
d	Actividad de disco
l (i)	Número de interrupciones/s
v	Utilización de tablas en el núcleo
y	Estadísticas de utilización de ttys
p	Información de paginación y actividad de <i>swap</i>
r	Memoria libre y ocupación de <i>swap</i>
l (L)	Estadísticas de red
L	Información de errores de red
w	Estadísticas de conexiones IP
t	Estadísticas de TCP
U	Estadísticas de UDP
m	Estadísticas de ICMP
N	Estadísticas de NFS
A	Todas las opciones

Entre `atsar` y `sar` solo existen algunas diferencias en cuanto a la manera de mostrar los datos y `sar` incluye unas cuantas opciones más (o diferentes). A continuación se verán algunos ejemplos de utilización de `sar` (exactamente igual que con `atsar`, solo puede haber alguna diferencia en la visualización de los datos) y el significado de la información que genera:

Utilización de CPU: `sar -u 4 5`

```
Linux debian 2.6.26-2-686 #1 SMP Thu May 28 15:39:35 UTC 2009 i686 11/30/2010
05:32:54 cpu %usr %nice %sys %irq %softirq %wait %idle _cpu_
05:33:05 all 3 0 8 0 0 88 0
05:33:09 all 4 0 12 0 0 84 0
05:33:14 all 15 0 19 1 0 65 0
...
05:41:09 all 0 0 1 0 0 0 99
```

`%usr` y `%sys` muestran el porcentaje de tiempo de CPU en el modo usuario con `nice=0` (normales) y en el modo núcleo. `idle` indica el tiempo no utili-

zado de CPU por los procesos en estado de espera (no incluye espera de disco). `wait` es el tiempo que la CPU ha estado libre cuando el sistema estaba realizando entrada o salida (por ejemplo de disco). `irq` y `sofirq` es el tiempo que la CPU ha dedicado a gestionar las interrupciones, que es un mecanismo de sincronización entre lo que hace la CPU y los dispositivos de entrada y salida. En el caso `idle=99%` significa que la CPU está ociosa, por lo que no hay procesos por ejecutar y la carga es baja; si `idle ≈` y el número de procesos es elevado, debería pensarse en optimizar la CPU, ya que podría ser el cuello de botella del sistema. En el ejemplo podemos ver que hay poca utilización de CPU y mucho uso de entrada y salida, por lo cual se puede verificar que en este caso la carga del sistema la está generando el disco (para el ejemplo se habían abierto 5 copias del programa OpenOffice Writer).

Número de interrupciones por segundo: `sar -I 4 5`

```
Linux debian 2.6.26-2-686 #1 SMP Thu May 28 15:39:35 UTC 2009 i686 11/30/2010
05:46:30 cpu iq00 iq01 iq05 iq08 iq09 iq10 iq11 iq12 iq14 iq15 _intr/s_
05:46:34 all 0 0 0 0 33 0 0 134 4 5
05:46:37 all 0 0 0 0 54 1 0 227 38 13
05:46:42 all 0 0 0 0 41 0 0 167 10 8
```

Muestra la información de la frecuencia de interrupciones de los niveles activos que se encuentran en `/proc/interrupts`. Nos es útil para ver si existe algún dispositivo que está interrumpiendo constantemente el trabajo de la CPU. Consultando este archivo veremos que en el ejemplo las más activas son la 9 (acpi), 12 (teclado), 14-15 (ide) y muy poco la 10 (usb).

Memoria y *swap*: `sar -r 4 5`

```
Linux debian 2.6.26-2-686 #1 SMP Thu May 28 15:39:35 UTC 2009 i686 11/30/2010
05:57:10 memtot memfree buffers cached slabmem swptot swpfree _mem_
05:57:17 1011M 317M 121M 350M 30M 729M 729M
05:57:21 1011M 306M 121M 351M 30M 729M 729M
05:57:25 1011M 300M 121M 351M 30M 729M 729M
```

En este caso `memtot` indica la memoria total libre y `memfree`, la memoria libre. El resto de indicadores es la memoria utilizada en `buffers`, la utilizada en caché (de datos), `slabmem` es la memoria dinámica del núcleo y `swptot/free` es el espacio total/libre de *swap*. Es importante tener en cuenta que si `memfree ≈ 0` (no hay espacio), las páginas de los procesos irán a parar al *swap*, donde debe haber sitio teniendo en cuenta que esto permitirá la ejecución, pero todo irá más lento. Esto se debe contrastar con el uso de CPU. También se debe controlar que el tamaño de los `buffers` sea adecuado y esté en relación con los procesos que están realizando operaciones de entrada y salida. Es también interesante el comando `free`, que permite ver la cantidad de memoria en una visión simplificada:

```

                total      used      free      shared  buffers   cached
Mem:           1036092    711940    324152         0     124256    359748
-/+ buffers/cache:    227936    808156
Swap:           746980         0     746980

```

Esto indica que de 1 Gb casi las 3/4 partes de la memoria están ocupadas y que aproximadamente 1/3 son de caché. Además, nos indica que el *swap* no se está utilizando para nada, por lo que podemos concluir que el sistema está bien. Si quisiéramos más detalles deberíamos utilizar el comando `vmstat` (con más detalles que el `sar -r`) para analizar qué es lo que está causando problemas o quién está consumiendo tanta memoria. A continuación se muestra una salida de `vmstat 1 10*`:

*Consúltese el manual para obtener una descripción de las columnas.

```

procs -----memory-----  ---swap--  -----io----  -system--  ----cpu----
 r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa
 1  1      0 324820 124256 359796  0  0  23  11  20 112  0  0 99  1
 0  0      0 324696 124256 359816  0  0   0  88   4  96  1  1 98  0
 0  0      0 324716 124256 359816  0  0   0   0 106 304  0  0 100  0
 0  0      0 324716 124256 359816  0  0   0   0 150 475  1  2 97  0
...

```

Utilización de las tablas del núcleo: `sar -v 4 5`

```

Linux  debian  2.6.26-2-686  #1 SMP Thu May 28 15:39:35 UTC 2009  i686  11/30/2010
06:14:02  superb-sz  inode-sz      file-sz      dquota-sz      flock-sz      _curmax_
06:14:06      0/0      32968/36      3616/101976      0/0      13/0
06:14:10      0/0      32968/36      3616/101976      0/0      13/0
06:14:13      0/0      32984/36      3616/101976      0/0      13/0
06:14:17      0/0      32984/36      3616/101976      0/0      13/0
06:14:22      0/0      33057/36      3680/101976      0/0      13/0

```

En este caso, `superb-sz` es el número actual-máximo de *superblocks* mantenido por el núcleo para los sistemas de archivos montados; `inode-sz` es el número actual-máximo de *incore-inodes* en el núcleo necesario, que es de uno por disco como mínimo; `file-sz` es el número actual-máximo de archivos abiertos, `dquota-sz` es la ocupación actual-máxima de entradas de cuotas (para más información consúltese `man sar -o atsar`). Esta monitorización se puede completar con el comando `ps -Af` (*process status*) y el comando `top`, que mostrarán la actividad y estado de los procesos en el sistema. A continuación, se muestran dos ejemplos de ambos comandos (solo algunas líneas):

```

debian:/proc# ps -Alw
F S  UID  PID  PPID  C  PRI  NI  ADDR  SZ  WCHAN  TTY      TIME  CMD
4 S   0    1    0  0  80   0  -    525  -    ?        00:00:01  init
5 S   0    2    0  0  75  -5  -     0  -    ?        00:00:00  kthreadd
1 S   0    3    2  0 -40  -  -     0  -    ?        00:00:00  migration/0
...
5 S   1  1601    1  0  80   0  -    473  -    ?        00:00:00  portmap
5 S  102  1612    1  0  80   0  -    489  -    ?        00:00:00  rpc.statd
...
4 S  113  2049  2012  0  80   0  -   31939  -    ?        00:00:03  mysqld
...
4 S   0  2654  2650  0  80   0  -    6134  -    tty7     00:00:49  Xorg
1 S   0  2726    1  0  80   0  -    6369  -    ?        00:00:00  apache2
0 S   0  2746    1  0  80   0  -     441  -    tty1     00:00:00  getty
...

```

Algunos aspectos interesantes para ver son la dependencia de los procesos (PPID=proceso padre) y, por ejemplo, que para saber el estado de los procesos se puede ejecutar con `ps -Alw` y en la segunda columna nos mostrará cómo se encuentra cada uno de los procesos. Estos parámetros reflejan el valor indicado en la variable del núcleo para este proceso, los más importantes de los cuales desde el punto de vista de la monitorización son: *F flags* (en este caso 1 es con superprivilegios, 4 creado desde el inicio *daemon*), *S* es el estado (D: no interrumpible durmiendo entrada/salida, R: ejecutable o en cola, S: durmiendo, T: en traza o parado, Z: muerto en vida, 'zombie'). *PRI* es la prioridad; *NI* es *nice*; *TTY*, desde dónde se ha ejecutado; *TIME*, el tiempo de CPU; *CMD*, el programa que se ha ejecutado y sus parámetros. Si se quiere salida con refresco (configurable), se puede utilizar el comando `top`, que muestra unas estadísticas generales (procesos, estados, carga, etc.) y, después, información de cada uno de ellos similar al `ps`, pero se actualiza cada 5 segundos por defecto:

```
top - 06:40:37 up 5:45, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 118 total, 2 running, 116 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.3%us, 4.1%sy, 0.0%ni, 93.2%id, 0.4%wa, 0.1%hi, 0.0%si, 0.0%st
Mem: 1036092k total, 722484k used, 313608k free, 124396k buffers
Swap: 746980k total, 0k used, 746980k free, 367408k cached
  PID USER  PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  COMMAND
 2654 root   20   0 41368  16m 8928 S  3.5  1.7   0:51.86 Xorg
 3093 remo   20   0 38764  15m 9532 R  1.8  1.6   0:22.24 gnome-terminal
 2987 remo   20   0 36924  19m 11m S  1.1  1.9   0:05.14 gnome-panel
 2984 remo   20   0 20604  11m 7252 S  0.7  1.2   0:03.86 metacity
 2980 remo   20   0 15464  5344 4336 S  0.6  0.5   0:05.34 gnome-screensav
 2988 remo   20   0 79820  28m 13m S  0.5  2.8   0:07.12 nautilus
 5263 remo   20   0 6036  3204 1312 S  0.4  0.3   0:00.08 bash
 2547 root   20   0 2052   868  736 S  0.1  0.1   0:01.16 dhcdbd
 2592 root   20   0 3388  1032  900 S  0.1  0.1   0:09.36 hald-addon-stor
```

También se pueden utilizar las herramientas del paquete *sysstat* para conocer el estado de los recursos, como por ejemplo `vmstat` (estadísticas de CPU, memoria y entrada/salida), `iostat` (estadísticas de discos y CPU) y `uptime` (carga de CPU y estado general).

1.1.2. Optimización del sistema

A continuación veremos algunas recomendaciones para optimizar el sistema en función de los datos obtenidos.

1) Resolver los problemas de memoria principal: Se debe procurar que la memoria principal pueda acoger un porcentaje elevado de procesos en ejecución, ya que si no es así, el sistema operativo podrá paginar e ir al *swap*; pero esto significa que la ejecución de ese proceso se degradará notablemente. Si se agrega memoria, el tiempo de respuesta mejorará notablemente. Para ello, se debe tener en cuenta el tamaño de los procesos (*SIZE*) en estado *R* y agregarle la que utiliza el núcleo. La cantidades de memoria se pueden ob-

tener con el comando `free`, que nos mostrará (o con `dmesg`), por ejemplo (`total/used/free/buffers/cached`):

```
1036092/723324/312768/124396/367472,
```

que es equivalente a (en megabytes) 1011/706/305/121/358 y donde observamos, en este caso, que solo el 30% de la memoria está libre y que en este momento no hay problemas, pero una carga mínima del sistema puede significar un problema. Es por ello que deberemos analizar si el sistema está limitado por la memoria (con `atsar -r y -p` se verá mucha actividad de paginación).

Las soluciones para la memoria son obvias: o se incrementa la capacidad o se reducen las necesidades. Por el coste actual de la memoria, es más adecuado incrementar su tamaño que emplear muchas horas para ganar un centenar de bytes al quitar, ordenar o reducir requerimientos de los procesos en su ejecución. Reducir los requerimientos puede hacerse reduciendo las tablas del núcleo, quitando módulos, limitando el número máximo de usuarios, reduciendo los *buffers*, etc.; todo lo cual degradará el sistema (efecto burbuja) y las prestaciones serán peores (en algunos casos, el sistema puede quedar totalmente no operativo).

Otro aspecto que se puede reducir es la cantidad de memoria de los usuarios gracias a la eliminación de procesos redundantes y cambiando la carga de trabajo. Para ello, se deberán monitorizar los procesos que están durmiendo (zombies) y eliminarlos, o bien aquellos que no progresan en su entrada/salida (saber si son procesos activos, cuánto de CPU llevan gastada y si los “usuarios están esperando por ellos”). Cambiar la carga de trabajo es utilizar planificación de colas para que los procesos que necesitan gran cantidad de memoria se puedan ejecutar en horas de poca actividad (por ejemplo, por la noche, lanzándolos con el comando `at`).

2) Mucha utilización de CPU: Básicamente nos la da el tiempo *idle* (valores bajos). Con `ps` o `top` se deben analizar qué procesos son los que “devoran CPU” y tomar decisiones, como posponer su ejecución, pararlos temporalmente, cambiar su prioridad (es la solución menos conflictivo de todas y para ello se puede utilizar el comando `renice prioridad PID`), optimizar el programa (para la próxima vez) o cambiar la CPU (o agregar otra). Como ya se ha mencionado, GNU/Linux utiliza el directorio `/proc` para mantener todas las variables de configuración del núcleo que pueden ser analizadas y, en cierto caso, “ajustadas”, para lograr prestaciones diferentes o mejores.

Para ello, se debe utilizar el comando `sysctl -a` para obtener todas las variables del núcleo y sus valores en el archivo*. Otros comandos alternativos son el `systemd` y `systemd-dump`, que permiten descargar las variables en un archivo y modificarlas, para cargarlas nuevamente en el `/proc` (el comando `systemd` guarda la configuración en `/etc/systemd.conf`). En este caso, por ejemplo,

*Consúltese el manual para cambiar los valores y el archivo de configuración `/etc/sysctl.conf`

se podrían modificar (se debe proceder con cuidado, porque el núcleo puede quedar fuera de servicio) las variables de la categoría `/proc/sys/vm` (memoria virtual) o `/proc/sys/kernel` (configuración del *core* del núcleo).

En este mismo sentido, también (para expertos o desesperados) se puede cambiar el tiempo máximo (*slice*) que el administrador de CPU (*scheduler*) del sistema operativo dedica a cada proceso en forma circular (si bien es aconsejable utilizar *renice* como práctica). Pero en GNU/Linux, a diferencia de otros sistemas operativos, es un valor fijo dentro del código, ya que está optimizado para diferentes funcionalidades (pero es posible tocarlo). Se puede “jugar” (a su propio riesgo) con un conjunto de variables que permiten tocar el *time slice* de asignación de CPU (`kernel-source-2.x.x/kernel/sched.c`).

3) Reducir el número de llamadas: Otra práctica adecuada para mejorar las prestaciones es reducir el número de llamadas al sistema de mayor coste en tiempo de CPU. Estas llamadas son las invocadas (generalmente) por el `shell fork()` y `exec()`. Una configuración inadecuada de la variable `PATH`, y debido a que la llamada `exec()` no guarda nada en caché, el directorio actual (indicado por `.`), puede tener una relación desfavorable de ejecución. Para ello, siempre habrá que configurar la variable `PATH` con el directorio actual como última ruta. Por ejemplo, en `$HOME/.bashrc` hacer: `PATH=$PATH:.; export PATH` si el directorio actual no está en el *path* o, si está, rehacer la variable `PATH` para ponerlo como última ruta.

Se debe tener en cuenta que una alta actividad de interrupciones puede afectar a las prestaciones de la CPU con relación a los procesos que ejecuta. Mediante monitorización (`atsar -I`) se puede mirar cuál es la relación de interrupciones por segundo y tomar decisiones con respecto a los dispositivos que las causan. Por ejemplo, cambiar de módem por otro más inteligente o cambiar la estructura de comunicaciones si detectamos una actividad elevada sobre el puerto serie donde se encuentra conectado.

4) Mucha utilización de disco: Después de la memoria, un tiempo de respuesta bajo puede ser debido al sistema de discos. En primer lugar, se debe verificar que se disponga de tiempo de CPU (por ejemplo, `idle >20%`) y que el número de entrada/salida sea elevado (por ejemplo, superior a 30 entrada/salida/s) utilizando `atsar -u` y `atsar -d`. Las soluciones pasan por:

- En un sistema multidisco, planificar dónde se encontrarán los archivos más utilizados para equilibrar el tráfico hacia ellos (por ejemplo `/home` en un disco y `/usr` en otro) y que puedan utilizar todas las capacidades de entrada/salida con caché y concurrente de GNU/Linux (incluso, por ejemplo, planificar sobre qué *bus ide* se colocan). Comprobar luego que existe un equilibrio del tráfico con `atsar -d` (o con `iostat`). En situaciones críticas se puede considerar la compra de un sistema de discos RAID que realizan este ajuste de forma automática.
- Tener en cuenta que se obtienen mejores prestaciones sobre dos discos pequeños que sobre uno grande del tamaño de los dos anteriores.

- En sistemas con un solo disco, generalmente se realizan, desde el punto de vista del espacio, cuatro particiones de la siguiente manera (desde fuera hacia dentro): `/`, `swap`, `/usr`, `/home`, pero esto genera pésimas respuestas de entrada/salida porque si, por ejemplo, un usuario compila desde su directorio `/home/user` y el compilador se encuentra en `/usr/bin`, la cabeza del disco se moverá a lo largo de toda su longitud. En este caso, es mejor unir las particiones `/usr` y `/home` en una sola (más grande), aunque puede representar algunos inconvenientes en cuanto a mantenimiento.
- Incrementar los *buffers* de caché de entrada/salida (véase, por ejemplo, `/proc/ide/hd...`).
- Si se utiliza un `extfs`, se puede usar el comando `dumpe2fs -h /dev/hdx` para obtener información sobre el disco y `tune2fs /dev/hdx` para cambiar algunos de los parámetros configurables del mismo.
- Obviamente, el cambio del disco por uno de mayor velocidad (mayores RPM) siempre tendrá un impacto positivo en un sistema limitado por la entrada/salida de disco [9].

5) Mejorar aspectos de TCP/IP: Examinar la red con el comando `atsar` (o también con `netstat -i` o con `netstat -s | more`) para analizar si existen paquetes fragmentados, errores, *drops*, desbordamientos, etc., que puedan estar afectando a las comunicaciones y, con ello, al sistema (por ejemplo, en un servidor de NFS, NIS, ftp o web). Si se detectan problemas, se debe analizar la red para considerar las siguientes actuaciones:

- Fragmentar la red mediante elementos activos que descarten paquetes con problemas o que no sean para máquinas del segmento.
- Planificar dónde estarán los servidores para reducir el tráfico hacia ellos y los tiempos de acceso.
- Ajustar parámetros del núcleo (`/proc/sys/net/`). Por ejemplo, para obtener mejoras en el *throughput* debemos ejecutar la siguiente instrucción:
`echo 600 >/proc/sys/net/core/netdev_max_backlog*`.

*Mínimo 300

6) Otras acciones sobre parámetros del núcleo: Existe otro conjunto de parámetros sobre el núcleo que es posible sintonizar para obtener mejores prestaciones, si bien, teniendo en cuenta lo que hemos tratado anteriormente, se debe ir con cuidado, ya que podríamos causar el efecto contrario o inutilizar el sistema. Consultad en la distribución del código fuente en el directorio `kernel-source-2.x/Documentation/sysctl` algunos archivos como por ejemplo `vm.txt`, `fs.txt` y `kernel.txt`. `/proc/sys/vm` controla la memoria virtual del sistema (*swap*) y permite que los procesos que no entran en la memoria principal sean aceptados por el sistema pero en el dispositivo de *swap*, por lo cual, el programador no tiene límite para el tamaño de su

programa (obviamente debe ser menor que el dispositivo de *swap*). Los parámetros susceptibles de sintonizar se pueden cambiar muy fácilmente con `sysctl` (o también con `gpowertweak`). `/proc/sys/fs` contiene parámetros que pueden ser ajustados de la interacción núcleo-sistema de ficheros, tal como `file-max` (y exactamente igual para el resto de los archivos de este directorio).

7) Generar el núcleo adecuado a nuestras necesidades: La optimización del núcleo significa escoger los parámetros de compilación de acuerdo a nuestras necesidades. Es muy importante primero leer el archivo `readme` del directorio `/usr/src/linux`.

Una buena configuración del núcleo permitirá que se ejecute más rápido, que se disponga de más memoria para los procesos de usuario y, además, resultará más estable. Hay dos formas de construir un núcleo: **monolítico** (mejores prestaciones) o **modular** (basado en módulos, que tendrá mejor portabilidad si tenemos un sistema muy heterogéneo y no se desea compilar un núcleo para cada uno de ellos). Para compilar su propio núcleo y adaptarlo a su hardware y necesidades, cada distribución tiene sus reglas (si bien el procedimiento es similar).

1.1.3. Optimizaciones de carácter general

Existen una serie de optimizaciones de índole general que pueden mejorar las prestaciones del sistema:

1) Bibliotecas estáticas o dinámicas: cuando se compila un programa, se puede hacer con una biblioteca estática (`libr.a`), cuyo código de función se incluye en el ejecutable, o con una dinámica (`libr.so.xx.x`), donde se carga la biblioteca en el momento de la ejecución. Si bien las primeras garantizan código portable y seguro, consumen más memoria. El programador deberá decidir cuál es la adecuada para su programa incluyendo `-static` en las opciones del compilador (no ponerlo significa dinámicas) o `-disable-shared`, cuando se utiliza el comando `configure`. Es recomendable utilizar (casi todas las distribuciones nuevas lo hacen) la biblioteca estándar `libc.a` y `libc.so` de versiones 2.2.x o superiores (conocida como Libc6) que reemplaza a las anteriores. En gcc 4.X por defecto se utilizan bibliotecas dinámicas, pero se puede forzar (no recomendado) a utilizar estáticas incluso para la `libc` (opciones `-static -static-libgcc` en contraposición con las por defecto `-shared -shared-libgcc`).

2) Selección del procesador adecuado: generar código ejecutable para la arquitectura sobre la cual correrán las aplicaciones. Algunos de los parámetros más influyentes del compilador son:

Enlaces de interés

Es interesante consultar los siguientes artículos:
http://people.redhat.com/alikins/system_tuning.html sobre información de optimización de sistemas servidores Linux y
<http://www.linuxjournal.com/article/2396>, *Performance Monitoring Tools for Linux*. El primero tiene ya más de cuatro años, pero la mayoría de los procedimientos continúan vigentes y el segundo es un artículo antiguo y algunas opciones no están disponibles, pero la metodología sigue siendo la misma.

- a) `-march` (por ejemplo, `-march=core2` para el soporte de CPU Intel Core2 CPU 64-bit con extensiones MMX, SSE, SSE2, SSE3/SSSE3, o `-march=k8` para CPU AMD K8 Core con soporte x86-64) haciendo simplemente `gcc -march=i686`;
- b) el atributo de optimización `-O1, 2, 3` (`-O3` generará la versión más rápida del programa, `gcc -O3 -march = i686`), y
- c) los atributos `-f` (consultad la documentación para los diferentes tipos).

3) Optimización del disco: en la actualidad, la mayoría de ordenadores incluye disco UltraDMA (100) por defecto; sin embargo, en una gran cantidad de casos no están optimizados para extraer las mejores prestaciones. Existe una herramienta (`hdparm`) que permite sintonizar el núcleo a los parámetros del disco tipo IDE y SATA (aunque estos últimos cuentan también con una utilidad específica llamada `sdparm`). Se debe tener cuidado con estas utilidades, sobre todo en discos UltraDMA (hay que verificar en el BIOS que los parámetros para soporte por DMA están habilitados), ya que pueden inutilizar el disco. Consultad las referencias y la documentación ([14] y `man hdparm/sdparm`) sobre cuáles son (y el riesgo que comportan) las optimizaciones más importantes, por ejemplo: `-c3`, `-d1`, `-X34`, `-X66`, `-X12`, `-X68`, `-mXX`, `-a16`, `-u1`, `-W1`, `-k1`, `-K1`. Cada opción significa una optimización y algunas son de altísimo riesgo, por lo que habrá que conocer muy bien el disco. Para consultar los parámetros optimizados, se podría utilizar `hdparm -vtT /dev/hdX` (donde `X` es el disco optimizado) y la llamada a `hdparm` con todos los parámetros se puede poner en `/etc/init.d` para cargarla en el *boot*. Para consultar la información del disco se puede hacer, por ejemplo, `hdparm -i /dev/sdb`

Paquetes no-free

Recordad que sobre Debian se debe activar el repositorio de paquetes `no-free` para poder instalar los paquetes de documentación del compilador `gcc-doc` y `gcc-doc-base`.

1.1.4. Configuraciones complementarias

Existen más configuraciones complementarias desde el punto de vista de la seguridad que de la optimización, pero son necesarias sobre todo cuando el sistema está conectado a una intranet o a Internet. Estas configuraciones implican las siguientes acciones [14]:

1) Impedir que se pueda arrancar otro sistema operativo: si alguien tiene acceso físico a la máquina, podría arrancar con otro sistema operativo preconfigurado y modificar el actual, por lo que se debe inhibir desde el BIOS del ordenador el *boot* por CD-ROM o USB y poner una contraseña de acceso (recordad la contraseña del BIOS, ya que, de otro modo, podría causar problemas cuando se quisiera cambiar la configuración).

2) Configuración y red: es recomendable desconectar la red siempre que se deseen hacer ajustes en el sistema. Se puede quitar el cable o deshabilitar el dispositivo con `/etc/init.d/networking stop` (`start` para activarla de nuevo) o con `ifdown eth0` (`ifup eth0` para habilitarla) para un dispositivo en concreto.

3) Modificar los archivos de `/etc/security`: de acuerdo a las necesidades de utilización y seguridad del sistema. En `access.conf` hay información sobre quién puede hacer un *login* al sistema; por ejemplo:

```
# Tabla de control de acceso. líneas con comuna=# es un comentario.
# El orden de la líneas es importante
# Formato: permission : users : origins
# Deshabilitar todo los logins excepto root sobre tty1
-:ALL EXCEPT root:tty1
# User "root" permitido conectarse desde estas direcciones
+ : root : 192.168.200.1 192.168.200.4 192.168.200.9
+ : root : 127.0.0.1
# O desde la red
+ : root : 192.168.201.
# Impide el acceso excepto user1,2,3 pero el último sólo desde consola.
-:ALL EXCEPT user1 user2 user3:console
```

También se debería, por ejemplo, configurar los grupos para controlar cómo y a dónde pueden acceder y también los límites máximos (`limits.conf`) para establecer los tiempos máximos de utilización de CPU, E/S, etc. y así evitar ataques por denegación de servicio (DoS).

4) Mantener la seguridad de la contraseña de root: utilizar como mínimo 6 caracteres, con uno, por lo menos, en mayúsculas o algún carácter que sea no trivial, como "-", ".", ",", etc.; asimismo, es recomendable activar el envejecimiento para forzar a cambiarlo periódicamente, así como también limitar el número de veces con contraseña incorrecta. También se puede cambiar el parámetro `min=x` de la entrada en `/etc/pam.d/passwd` para indicar el número mínimo de caracteres que se utilizarán en la contraseña (`x` es el número de caracteres).

5) No acceder al sistema como root: si bien muchas distribuciones ya incorporan un mecanismo de este estilo (por ejemplo, Ubuntu), se puede crear una cuenta como `sysadm` y trabajar con ella. Si se accede remotamente, habrá siempre que utilizar `ssh` para conectarse al `sysadm` y, en caso de ser necesario, realizar un `su -` para trabajar como `root` o activar el `sudoers` para trabajar con el comando `sudo` (consultad la documentación para las opciones del comando y su edición)

6) Tiempo máximo de inactividad: inicializar la variable `TMOUT`, por ejemplo a 360 (valor expresado en segundos), que será el tiempo máximo de inactividad que esperará el *shell* antes de bloquearse; se puede poner en los archivos de configuración del *shell* (por ejemplo, `/etc/profile`, `.profile`, `$HOME/.bashrc`, etc.). En caso de utilizar entornos gráficos (KDE, Gnome, etc.), se puede activar el salvapantallas con contraseña, al igual que el modo de suspensión o hibernación.

7) Configuración del NFS en forma restrictiva: en el `/etc/exports`, exportar solo lo necesario, no utilizar comodines (*wildcards*), permitir solo el acceso de lectura y no permitir el acceso de escritura por `root`, por ejemplo, con `/directorio_exportado host.domain.com (ro, root_squash)`.

8) Evitar arranques desde el *bootloader* con parámetros: se puede iniciar el sistema como *linux single*, lo que arrancará el sistema operativo en modo de

usuario único. hay que configurar el sistema para que el arranque de este modo siempre sea con contraseña. Para ello, en el archivo `/etc/inittab` se debe verificar que existe la línea `S:wait:/sbin/sulogin` y que tiene habilitado el `/bin/sulogin`. Por ejemplo, si trabajamos con Lilo como *bootloader*, el archivo `/etc/lilo.conf` debe tener los permisos adecuados para que nadie lo pueda modificar excepto el root (`chmod 600 /etc/lilo.conf`). Para prevenir cambios accidentales, se debe cambiar el atributo de bloqueo con `chattr +i /etc/lilo.conf` (`usad -i` cuando se desee cambiar). Este archivo permite una serie de opciones que es conveniente considerar: `timeout=0` si el sistema tiene solo un sistema operativo para que haga el *boot* inmediatamente; `restricted` para evitar que se puedan insertar comandos en el momento del *boot* como `linux init = /bin/sh` y tener acceso como root sin autorización; en este caso, debe acompañarse de `password=palabra-de-password`; si solo se pone `password`, solicitará la contraseña para cargar la imagen del núcleo. Consultad Grub(2), que tiene opciones similares.

Se puede probar el riesgo que esto representa haciendo lo siguiente (siempre que el Grub/Lilo no tenga contraseña), que puede ser útil para entrar en el sistema cuando no se recuerda la contraseña de usuario, pero representa un gran peligro de seguridad cuando es un sistema y se tiene acceso a la consola y el teclado:

- a) se arranca el ordenador hasta que se muestre el menú de arranque,
- b) se selecciona el núcleo que se desea arrancar y se edita la línea presionando la tecla "e" (edit),
- c) buscamos la línea que comienza por `kernel . . .` y presionamos nuevamente la tecla "e",
- d) al final de la línea borramos el parámetro `ro` e introducimos `rw init=/bin/bash` (lo cual indica acceso directo a la consola)
- e) presionamos "enter" y finalmente "b" (boot). Arrancaremos el sistema y pasaremos directamente a modo root, gracias a lo cual se podrá cambiar la contraseña (incluida la de root), editar el fichero `/etc/passwd` o el `/etc/shadow` o también crear un nuevo usuario y todo lo que deseemos.

9) Control de la combinación *Ctrl-Alt-Delete*: Para evitar que se apage la máquina desde el teclado, se debe insertar un comentario (#) en la primera columna de la línea siguiente: `ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now` del archivo `/etc/inittab`. Los cambios se activan con la orden `telinit q`.

10) Evitar peticiones de servicios no ofrecidos: se debe bloquear el archivo `/etc/services`, para no admitir servicios no contemplados, por medio de `chattr +i /etc/services`.

11) Conexión del root: hay que modificar el archivo `/etc/securetty` que contiene las TTY y VC (*virtual console*) en que se puede conectar el root dejando solo una de cada, por ejemplo, `ttY1` y `vc/1` y, si es necesario, hay que conectarse como `sysadm` y hacer un `su`.

12) Eliminar usuarios no utilizados: se deben borrar los usuarios o grupos que no sean necesarios, incluidos los que vienen por defecto (por ejemplo, `operator`, `shutdown`, `ftp`, `uucp`, `games`, etc.) y dejar solo los necesarios (`root`,

bin, daemon, sync, nobody, sysadm) y los que se hayan creado con la instalación de paquetes o por comandos (lo mismo con `/etc/group`). Si el sistema es crítico, podría considerarse el bloqueo (`chattr +i file`) de los archivos `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/gshadow` para evitar su modificación (cuidado con esta acción, porque no permitirá cambiar posteriormente las contraseñas).

13) Montar las particiones en forma restrictiva: utilizar en `/etc/fstab` atributos para las particiones tales como `nosuid` (que impide suplantar el usuario o grupo sobre la partición), `nodev` (que no interpreta dispositivos de caracteres o bloques sobre esta partición) y `noexec` (que no permite la ejecución de archivos sobre esta partición). Por ejemplo: `/tmp /tmp ext2 defaults,nosuid,noexec 0 0`. También es aconsejable montar el `/boot` en una partición separada y con atributos `ro`.

14) Protecciones varias: se puede cambiar a 700 las protecciones de los archivos de `/etc/init.d` (servicios del sistema) para que solo el root pueda modificarlos, arrancarlos o pararlos y modificar los archivos `/etc/issue` y `/etc/issue.net` para que no den información (sistema operativo, versión, etc.) cuando alguien se conecta por telnet, ssh, etc.

15) SUID y SGID: un usuario podrá ejecutar como propietario un comando si tiene el bit SUID o SGID activado, lo cual se refleja como una 's' SUID (`-rwsr-xr-x`) y SGID (`-r-xr-sr-x`). Por lo tanto, es necesario quitar el bit (`chmod a-s file`) a los comandos que no lo necesitan. Estos archivos pueden buscarse con: `find / -type f -perm -4000 o -perm -2000 -print`. Se debe proceder con cuidado respecto a los archivos en que se quite el SUID-GUID, porque el comando podría quedar inutilizado.

16) Archivos sospechosos: hay que buscar periódicamente archivos con nombres no usuales, ocultos o sin un uid/gid válido, como `"..."` (tres puntos), `".. "` (punto punto espacio), `"..G"` o equivalentes. Para ello, habrá que utilizar: `find / -name=".*" -print | cat -v` o sino `find / -name "..." -print`.

Para buscar uid/gid no válidos, utilizad `find / -nouser` (o utilizad también `-nogroup` (cuidado, porque algunas instalaciones se hacen con un usuario que luego no está definido y que el administrador debe cambiar).

17) Conexión sin contraseña: no se debe permitir el archivo `.rhosts` en ningún usuario, a no ser que sea estrictamente necesario (se recomienda utilizar ssh con clave pública en lugar de métodos basados en `.rhosts`).

18) X Display manager: para indicar los `hosts` que se podrán conectar a través de XDM y evitar que cualquier `host` pueda tener una pantalla de `login` se puede modificar el archivo `/etc/X11/xdm/Xaccess`.

1.1.5. Resumen de acciones para mejorar un sistema

1) Observar el estado del sistema y analizar los procesos que consumen mucha CPU utilizando, por ejemplo, el comando `ps auxS -H` (o el comando

top) y mirando las columnas %CPU, %MEM y TIME; se debe observar la jerarquía de procesos y prestar atención a cómo se está utilizando la CPU y la memoria y analizar el tiempo de ejecución de los procesos para encontrar procesos *zombies* mirando en la columna STAT aquellos que tengan el identificador Z (los cuales se podrán eliminar sin problemas). También se debe prestar especial atención a los que estén con D, S (que están haciendo entrada o salida) y W (que están utilizando el *swap*). En estos tres últimos utilizad el *atsar* y *free* (o *vmstat*) para verificar la carga de entrada y salida, ya que puede ser que estos procesos estén haciendo que las prestaciones del sistema bajen notablemente (generalmente por sus necesidades, en cuyo caso no podremos hacer gran cosa, pero en otros casos puede ser que el código no esté optimizado o bien escrito).

2) Analizar el estado de la memoria en detalle para descubrir dónde se está gastando la memoria. Recordad que todos los procesos que se deben ejecutar deben estar en memoria principal y, si no hay, el proceso paginará en *swap* pero con la consiguiente pérdida de prestaciones, ya que debe ir al disco y llevar zona de memoria principal. Es vital que los procesos más activos tengan memoria principal y esto se puede lograr cambiando el orden de ejecución o haciendo un cambio de prioridades (comando *renice*). Para observar el estado de la memoria en detalle utilizad el *vmstat 2* (o el *atsar*), por ejemplo, y observad las columnas *swpd*, que es la cantidad de memoria virtual (*swap*) utilizada, *free*, la cantidad de memoria principal libre (la ocupada se obtiene de la total menos la libre) y *si/so*, la cantidad de memoria virtual en lectura o escritura utilizada. Si tenemos un proceso que utiliza gran cantidad de *swap* (*si/so*) este proceso estará gastando mucho tiempo en gestión, retardará el conjunto y veremos que la CPU tiene, por ejemplo, valores de utilización bajos. En este caso se deberían eliminar procesos de la memoria para hacer sitio o ampliar la memoria RAM del sistema si es que no se pueden quitar los procesos que hay, siempre y cuando el proceso bajo estudio no sea de ejecución ocasional.

3) Tened en cuenta que %CPU + %E/S + %Idle = 100% por lo cual vemos que la E/S (I/O en *vmstat*) también afecta a un proceso.

```

debian:/home/remo# vmstat 2
procs -----memory----- --swap-- -----io----- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo    in   cs  us  sy  id  wa
...
 0  0     0 623624 29400 231596    0    0  7184     0  393 1389 10  8 10 73
 0  0     0 623596 29400 231612    0    0     0     0  416  800  0  2 98  0
 1  0     0 622540 29408 231604    0    0     0  276  212  549  2  2 94  2
 0  0     0 613544 29536 240620    0    0  4538     0  464 1597 10  8 29 54
 0  0     0 612552 29560 240824    0    0   112     0  412  850  1  2 85 12

```

En este caso podemos observar que hay una utilización muy grande de I/O (E/S) pero 0 de *swap* un y alto valor de CPU tanto en *wa* (*waiting*) como en *id* (*idle*) lo que quiere decir que la CPU está esperando a que algo que está en E/S, termine (en este caso es la ejecución de unas cuantas instancias del OpenOffice, lo que significa lectura de disco y carga de un ejecutable a memoria principal). Si esta situación se repite o es constante, se debería analizar

cómo utilizan la memoria los procesos en espera de ejecución y cómo reducirla (por ejemplo, poniendo un disco más rápido o con más *buffer* de E/S).

4) Se debe tener en cuenta que el %CPU está constituido por la suma de dos valores “us” (User Time) y “sy” (System Time). Estos valores representan el tiempo empleado ejecutando código del usuario (*non-kernel code*) y el tiempo gastado ejecutando código del núcleo, respectivamente y pueden ser útiles cuando se desea optimizar el código de un programa con el fin de que consuma menos tiempo de CPU. Utilizaremos el comando `time`, que nos da el tiempo gastado en cada tipo de código, haciendo, por ejemplo, `time find /usr`, de modo que tendremos que nos da `real 1m41.010s, user 0m0.076s, sys 0m2.404s`; en cambio, si hacemos `time ls -R /usr` la salida es `real 0m5.530s user 0m0.160s sys 0m0.068s`. Como vemos, para la obtención de información equivalente (listado de archivos y directorios) un comando ha gastado 2,404 s en espacio de núcleo y el otro 0,06 s, por lo cual es interesante analizar qué comandos escogemos para hacer el trabajo. Otro aspecto interesante es que si ejecutamos, por ejemplo, `time find /var >/dev/null` (para no ver la salida) la primera vez obtenemos `real 0m23.900s, user 0m0.000s, sys 0m0.484s` pero una segunda vez obtenemos `real 0m0.074s, user 0m0.036s, sys 0m0.036s`. ¿Qué ha pasado? El sistema ha almacenado en las tablas de caché la información y la siguientes veces ya no tarda lo mismo, sino mucho menos. Si se desea utilizar el `time` en el formato avanzado o extendido, los usuarios que ejecuten *bash* como *shell* deberán ejecutar el `time` junto con el *path* donde se encuentre; por ejemplo `/usr/bin/time ls -R /usr`, para obtener los resultados deseados (consultad `man time` para más información).

5) Es interesante ver qué optimizaciones podemos generar en un código con modificaciones simples. Por ejemplo, observemos el código desarrollado por Bravo [1]:

```
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
int main(void)
{int x=1, y=2, z=3; long iter1=0,iter2=0;
 struct timeval tv1,tv2;
 gettimeofday(&tv1,NULL);
 for(;;){
     x=(x*3+y*7+z*9)%11;
     y=(x*9+y*11+z*3)%29;
     z=(x*17+y*13+z*11)%37;
     iter1++;
     if(iter1==1000000){ iter2++; iter1=0;}
     gettimeofday(&tv2,NULL);
     if(tv2.tv_sec==tv1.tv_sec+5 && tv2.tv_usec>=tv1.tv_usec || tv2.tv_sec>tv1.tv_sec+5)
     break;}
 printf("Iteraciones: %ldM Resultado: %d %d %d\n",iter2,x,y,z);
 return 0;}
```

El resultado de la ejecución es

```
time ./c:Iteraciones: 22M real 0m5.001s, user 0m1.756s, sys 0m3.240s
```



donde se puede observar que los 3,240 s han alcanzado para 22 millones de iteraciones. ¿En qué se gastan los 3,240 s? Pues en calcular la hora en cada iteración, ya que son múltiples llamadas al núcleo. Si mejoramos el código y solo calculamos el `gettimeofday` cada millón de iteraciones, obtenemos Iteraciones: 135M real 0m5.025s, user 0m4.968s, sys 0m0.056s y vemos que se reduce notablemente el tiempo `sys` y obtenemos más tiempo para ejecutar el código del usuario, por lo cual sube el número de iteraciones (135 millones), teniendo en cuenta que hemos pasado de 22 millones de ejecución de la llamada `gettimeofday` a 135 millones de veces. ¿Cuál es la consecuencia? Que la finalización de ejecución de este ejemplo se obtiene por comparación de 5 s con el tiempo absoluto, por lo cual al calcular el tiempo absoluto menos veces se obtiene cierta “imprecisión” al determinar cuándo finaliza la ejecución (`real 0m5.001s` en el primer caso, mientras que en el segundo `0m5.025s`, una diferencia de 24 milésimas). Una solución optimizada para este caso sería no usar este tipo de llamadas al sistema para determinar cuándo debe finalizar un proceso y buscar alternativas, por ejemplo, con `alarm` y una llamada a `signal`. [1]

1.2. Monitorización

Un aspecto importante en el funcionamiento 24x7 de un sistema es que el administrador se debe anticipar a los problemas y es por ello que o bien está continuamente mirando su funcionamiento (lo cual es prácticamente imposible todo el tiempo) o bien se dispone de herramientas adecuadas que puedan prevenir la situación, generar alertas y advertir al responsable de que “algo está pasando” y que este pueda realizar con antelación las acciones correctivas para evitar el fallo, disfunción o situación de fuera de servicio del sistema o recurso. Las herramientas que cumplen esta función se enmarcan dentro del grupo de herramientas de monitorización y permiten también obtener información del sistema con fines estadísticos, contables u otros que el usuario desee. Las herramientas más comunes permiten, mediante una interfaz web, conocer de forma remota los cinco factores principales (uso de CPU, memoria, E/S, red, procesos/servicios) que dan indicios de que “alguna cosa puede estar pasando”; las más sofisticadas generan alarmas por SMS para advertir de la situación al administrador. A continuación se describirán algunas de las herramientas más representativas (pero no son las únicas): Munin, Monit, MRTG y Cacti.

1.2.1. Munin

Munin [16] produce gráficos sobre diferentes parámetros del servidor (load average, memory usage, CPU usage, MySQL throughput, eth0 traffic, etc.) sin excesivas configuraciones y presenta gráficos importantes para reconocer dónde y qué está generando problemas. Consideremos que nuestro sis-



Cacti se describe en el módulo “Clusterización” junto con Ganglia, ya que están más orientadas a clústers de ordenadores.

tema se llama `debian.nteum.org` y tenemos nuestra página configurada como `debian.nteum.org` con los documentos en `/var/www/`. Para instalar Munin sobre Debian hacemos, por ejemplo, `apt-get install munin munin-node`. Luego debemos configurar Munin (`/etc/munin/munin.conf`) con:

```
dbdir /var/lib/munin
htmldir /var/www/munin
logdir /var/log/munin
rundir /var/run/munin
tmpldir /etc/munin/templates
[debian.nteum.org]
  address 127.0.0.1
  use_node_name yes
```

Luego se crea el directorio, se cambian los permisos y se reinicia el servicio (caso de no existir).

```
mkdir -p /var/www/munin
chown munin:munin /var/www/munin
/etc/init.d/munin-node restart
```

Después de unos minutos se podrán ver los primeros resultados en la dirección web `http://debian.nteum.org/munin` o en `http://localhost/munin` en el navegador. Si se quiere mantener la privacidad de las gráficas basta con poner una contraseña para el acceso con Apache al directorio. Por ejemplo, se pone en el directorio `/var/www/munin/` el archivo `.htaccess` con el siguiente contenido:

```
AuthType Basic
AuthName "Members Only"
AuthUserFile /var/www/munin/.htpasswd
<limit GET PUT POST>
  require valid-user
</limit>
```

Luego se debe crear el archivo de contraseña en `/var/www/munin/.htpasswd` con el comando (como root): `htpasswd -c /var/www/munin/.htpasswd admin`. Cuando nos conectemos al `http://localhost/munin/` nos pedirá el usuario (admin) y la contraseña que hemos introducido después del comando anterior.

1.2.2. Monit

Monit [10] permite configurar y verificar la disponibilidad de servicios tales como Apache, MySQL o Postfix y toma diferentes acciones, como por ejemplo reactivarlos si no están presentes. Para instalar Monit hacemos `apt-get install monit` y editamos `/etc/monit/monitrc`. El archivo por defecto incluye un conjunto de ejemplos, pero se deberá consultar la documentación para obtener más información*. A continuación presentamos un ejemplo de configuración típico sobre algunos servicios `/etc/monit/monitrc` [11]:

*<http://mmonit.com/monit>

```

# Monit control file example: /etc/monit/monitrc
#
set daemon 120 # Poll at 2-minute intervals
set logfile syslog facility log_daemon
set alert root@debian.nteum.org
# Set a default mail from-address for all alert messages emitted by monit.
# All alert mail will be sent to below mail address.
set mail-format { from: root@debian.nteum.org }
set alert root@debian.nteum.org
# Make monit start its web-server. So you can access it from webbrowser.
set httpd port 2812 and
use address localhost
##Monit web-server ACL.
allow localhost # allow localhost to connect to the server and
#allow 192.168.1.2 # allow 192.168.1.2 to connect to the server,
# You can give only one per entry
allow admin:monit # user name and password for authentication.
allow hardik:hardik # set multiple user to access through browser.
# Monitoring the apache2 web services. It will check process apache2 with given pid file.
# If process name or pidfile path is wrong then monit will
# give the error of failed. tough apache2 is running.
check process apache2 with pidfile /var/run/apache2.pid
#Below is actions taken by monit when service got stuck.
start program = "/etc/init.d/apache2 start"
stop program = "/etc/init.d/apache2 stop"
# Admin will notify by mail if below of the condition satisfied.
if cpu is greater than 60% for 2 cycles then alert
if cpu > 80% for 5 cycles then restart
if totalmem > 200.0 MB for 5 cycles then restart
if children > 250 then restart
if loadavg(5min) greater than 10 for 8 cycles then stop
if 3 restarts within 5 cycles then timeout
group server
check process sshd with pidfile /var/run/sshd.pid
start program = "/etc/init.d/ssh start"
stop program = "/etc/init.d/ssh stop"
if failed port 22 protocol ssh then restart
if 5 restarts within 5 cycles then timeout
check process mysql with pidfile /var/run/mysqld/mysqld.pid
group database
start program = "/etc/init.d/mysql start"
stop program = "/etc/init.d/mysql stop"
if failed host 127.0.0.1 port 3306 then restart
if 5 restarts within 5 cycles then timeout
#Check host for which services up/down on particular port.
check host debian.nteum.org with address 10.0.2.15
if failed icmp type echo with timeout 4 seconds then alert
if failed port 21 then alert
if failed port 80 protocol http then alert
# if failed port 389 type tcp with timeout 15 seconds then alert
#include /etc/monit/default.monitrc
#include /etc/monit/mysql.monitrc

```

Enlace de interés

Consultad el manual para obtener más detalles en <http://mmonit.com/monit>.

Para verificar que la sintaxis es correcta ejecutamos `monit -t` y para ponerlo en marcha ejecutamos `monit`. A partir de este momento se puede consultar en la dirección y puerto seleccionado en el archivo `/etc/monit/monitrc` (en nuestro caso `http://localhost:2812/`), que nos pedirá el usuario y contraseña también introducidos en el mismo archivo (`admin` y `monit` en nuestro caso). Se puede parar y arrancar el servicio con `/etc/init.d/monit stop|start`, si bien para que nos permita hacer el arranque, deberemos cambiar en `/etc/default/monit` la variable `startup=1`.

1.2.3. MRTG

El MRTG (*Multi-Router Traffic Grapher*) [15] fue creado para mostrar información gráfica sobre datos de red, como se verá en el ejemplo que mostramos a continuación, para monitorizar la red Ethernet, pero se pueden usar otros datos para visualizar el comportamiento y para generar las estadísticas de carga (*load average*) del servidor. Para ello hacemos uso de los paquetes `mrtg` (`apt-get install mrtg mrtg-contrib mrtgutils`) y `atsar`. Una vez instalados, configuramos el fichero `/etc/mrtg.cfg` [5].

```
# Multi Router Traffic Grapher -- Sample Configuration File
# This file is for use with mrtg-2.5.4c

# Global configuration
  WorkDir: /var/www/mrtg
  WriteExpires: Yes
  Title[^]: Traffic Analysis for

# 128K leased line
# -----
#Title[leased]: a 128K leased line
#PageTop[leased]: <H1>Our 128K link to the outside world</H1>
#Target[leased]: 1:public@router.localnet
#MaxBytes[leased]: 16000

  Target[eth1]: `/usr/bin/mrtg-ip-acct eth1`
  MaxBytes1[eth1]: 32000
  MaxBytes2[eth1]: 16000
  Title[eth1]: Packets from eth1
  YLegend[eth1]: Traffic
  PageTop[eth1]: <H3>Analysis of total Internet traffic</H3>

  Target[average]: `/usr/bin/cpu-load-mrtg`
  MaxBytes[average]: 1000
  Options[average]: gauge, nopercent, growright, integer
  YLegend[average]: Load average
  kMG[average]: ,,
  ShortLegend[average]:
  Legend1[average]: Load average x 100
  LegendI[average]: load:
  LegendO[average]:
  Title[average]: Load average x 100 for debian.nteum.org
  PageTop[average]: <H3>Load average x 100 for debian.nteum.org</H3>
  <TABLE>
    <TR><TD>System:</TD>
      <TD>debian.nteum.org</TD></TR>
    <TR><TD>Maintainer:</TD> <TD>webmaster@debian.nteum.org</TD></TR>
    <TR><TD>Max used:</TD> <TD>1000</TD></TR>
  </TABLE>
```

Para generar los datos con `atsar` (o `sar`) creamos un *script* en el directorio `/usr/local/bin/cpu-load/average` (que tenga permisos de ejecución para todos) que pasará los datos a `mrtg`:

```
#!/bin/sh
load=`/usr/bin/atsar -u 1 | tail -n 1 | awk -F" " '{print $9}'`
echo "($load -100) * 100 " | bc | awk -F"." '{print $1}'
```

Deberemos crear y cambiar los permisos del directorio `/var/www/mrtg`. Por defecto, `mrtg` se ejecuta en el `cron`, pero si después queremos ejecutarlo, podemos hacer `mrtg /etc/mrtg.cfg` y esto generará las gráficas en el archivo `/var/www/mrtg/average.html`, que podremos visualizar con un navegador desde `http://debian.nteum.org/mrtg/average.html` o también desde `http://localhost/mrtg/average.html`. Por otro lado, la gráfica de carga de CPU o la de carga de red la podremos consultar en la siguiente dirección: `http://debian.nteum.org/mrtg/eth1.html` o también desde la dirección `http://localhost/mrtg/eth1.html`. A continuación tenéis una posible ejecución de `mrtg` a través del `cron` en el archivo `/etc/cron.d/mrtg`:

```
* /5 * * * * root if [ -x /usr/bin/mrtg ] && [ -r /etc/mrtg.cfg ];
then env LANG=C /usr/bin/mrtg /etc/mrtg.cfg 2>&1 | tee -a /var/log/mrtg/mrtg.log ; fi
```

Enlace de interés

Existen herramientas más sofisticadas para la monitorización de red y servicios de red utilizando SNMP (*Simple Network Management Protocol*) y MRTG (*Multi-Router Traffic Grapher*), por ejemplo. Más información al respecto se puede encontrar en la siguiente dirección: `http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_-_Ch22_-_Monitoring_Server_Performance`.

1.2.4. Otras herramientas

Otros paquetes interesantes a tener en cuenta para monitorizar un sistema son los siguientes:

- **Nagios***: Nagios es un sistema de monitorización de redes de código abierto ampliamente utilizado, que permite monitorizar tanto el hardware como los servicios que se especifiquen, generando alertas cuando el comportamiento de los mismos no sea el deseado. Permite la monitorización de servicios de red (SMTP, POP3, HTTP, SNMP, etc.), de recursos de sistemas de hardware (carga del procesador, uso de los discos, memoria, estado de los puertos, etc.), tiene posibilidad de monitorización remota mediante túneles SSL cifrados o SSH y la posibilidad de programar conectores (*plug-ins*) específicos para nuevos sistemas.
- **Cacti***: Cacti es una solución gráfica diseñada para trabajar conjuntamente con datos de RRDTool. Cacti provee diferentes formas de gráficas, métodos de adquisición y características que puede controlar el usuario muy fácilmente y es una solución que se adapta desde una máquina a un entorno complejo de máquinas, redes y servidores.
- **Frysk***: el objetivo del proyecto Frysk es crear un sistema de monitorización distribuido e inteligente para monitorizar procesos e hilos.

*<http://www.nagios.org>

Enlace de interés

Podéis consultar la guía de instalación rápida de Nagios para Debian en: `http://debianclusters.org/index.php/Nagios_Installation_and_Configuration [6]`.

*<http://cacti.net>

Cacti se describe en el módulo "Clusterización".



*<http://sources.redhat.com/frysk>

A continuación se describirán otras herramientas no menos interesantes (por orden alfabético) que incorpora GNU/Linux (por ejemplo, Debian) para la

monitorización de sistema. No es una lista exhaustiva, sino una selección de las más utilizadas (se recomienda ver el man de cada herramienta para mayor información):

- **atsar, ac, sac, sysstat, isag:** herramientas de auditoría tales como *ac*, *last*, *accton*, *sa*, *atsar* o *isag* (Interactive System Activity Grapher) para la auditoría de recursos hw y sw.
- **arpwatch; mon:** monitor de actividad Ethernet/FDDI que indica si hay cambios en tablas MACIP; monitor de servicios de red.
- **diffmon, fcheck:** generación de informes sobre cambios en la configuración del sistema y monitorización del sistemas de ficheros para detectar intrusiones.
- **fam:** file alteration monitor, monitor de alteración de ficheros.
- **genpower:** monitor para gestionar los fallos de alimentación.
- **gkrellm:** monitorización gráfica de CPU, procesos (memoria), sistemas de ficheros y usuarios, disco, red, Internet, *swap*, etc.
- **ksensors (lm-sensors):** monitor de la placa base (temperatura, alimentación, ventiladores, etc.).
- **lcap, systune:** herramienta para retirar capacidades asignadas al núcleo en el fichero `/proc/sys/kernel` y se adapta según las necesidades con *systune*.
- **powertweak:** monitorización y modificación de diferentes parámetros del hardware, núcleo, red, VFS o VM (permite modificar algunos de los parámetros mostrados anteriormente sobre `/proc`).
- **gps, gtop, lavaps (de más a menos amigable):** monitores de procesos de varios tipos (generalmente utilizan información de `/proc`) y permiten ver recursos, zócalos (*sockets*), archivos, entorno y otra información que estos utilizan, así como administrar sus recursos y estados.
- **swatch:** monitor para la actividad del sistema a través de archivos de registro.
- **vtgrab:** monitorización de máquinas remotas (similar a VNC).
- **whowatch:** herramienta en tiempo real para la monitorización de usuarios.
- **wmnd:** monitor de tráfico de red y monitorización de un clúster por red.
- **xosview:** monitor de recursos gráfico.

1.3. Alta disponibilidad en Linux (High-Availability Linux)

Actualmente Linux es conocido como un sistema operativo estable; los problemas se generan cuando el hardware falla. En los casos en que un fallo de hardware provoca graves consecuencias, debido a la naturaleza del servicio (aplicaciones críticas), se implementan sistemas tolerantes a fallos (*fault tolerant*, FT) con los cuales se garantiza, con una determinada probabilidad (muy alta), que el servicio esté siempre activo. El problema de estos sistemas es que son extremadamente caros, suelen ser soluciones cerradas, totalmente depen-

dientes de la solución integrada. Los sistemas de alta disponibilidad (*high availability*, HA) intentan obtener prestaciones cercanas a la tolerancia a fallos, pero a costes accesibles. La alta disponibilidad está basada en la replicación de elementos, por lo cual dejaremos de tener un servidor y necesitaremos tener un clúster de alta disponibilidad. Existen para Linux diferentes soluciones, como por ejemplo Heartbeat (elemento principal del Linux-HA), ldirectord y LVS (Linux Virtual Server), Piranha (solución basada en LVS de Red Hat), UltraMonkey (solución de VA Linux), Kimberlite (de Mission Critical Linux) o OpenAIS+Corosync+Pacemaker.

El proyecto Linux-HA (Linux de alta disponibilidad) [8] es una solución clúster de alta disponibilidad para Linux y otros sistemas operativos, como FreeBSD, OpenBSD, Solaris y MacOSX y que provee fiabilidad, disponibilidad y prestación discontinua de servicios. El producto principal del proyecto es **Heartbeat**, cuyo objetivo principal es la gestión de clústers con el objetivo de obtener alta disponibilidad. Sus más importantes características son: ilimitado número de nodos (útil tanto para pequeños clústers como para tamaños grandes), monitorización de recursos (estos se pueden reiniciar o desplazar a otro nodo en caso de fallo), mecanismo de búsqueda para eliminar nodos con fallos del clúster, gestión de recursos basada en directivas o reglas con posibilidad de incluir el tiempo, gestión preconfigurada de recursos (Apache, DB2, Oracle, PostgreSQL, etc.) e interfaz gráfica de configuración. Para poder ser útiles a los usuarios, el daemon de Heartbeat tiene que combinarse con un administrador de recursos de clúster (CRM), que tiene la tarea de iniciar y detener los servicios (direcciones IP, servidores web, etc.) lo cual proporcionará la alta disponibilidad. Desde la versión 2.1.3 de Heartbeat se ha sustituido el código del gestor de recursos del clúster (CRM) por el componente Pacemaker. Pacemaker logra la máxima disponibilidad de sus servicios de clúster mediante la detección y recuperación de nodos y los fallos de nivel de servicio. Esto se logra mediante la utilización de las capacidades de mensajería y la pertenencia a la infraestructura proporcionada OpenAIS o Pacemaker.

1.3.1. Guía breve de instalación de Heartbeat en Debian

En este subapartado se dará una breve descripción de cómo construir un clúster de alta disponibilidad de dos nodos con Heartbeat*. Es interesante (aunque un poco antiguo) el artículo [7] y la documentación del sitio web de Linux-HA [3]. Como punto inicial disponemos de dos ordenadores similares (no es necesario, pero si uno debe ocupar el lugar del otro es aconsejable). A los servidores los llamaremos NteumA (primario) y NeteumB (secundario), con dos interfaces de red cada uno: la primera para que se comuniquen entre si y la otra servirá a cada uno de ellos para conectarse a Internet, por ejemplo. De esta forma NteumA: eth0 = 10.0.2.20 y NeteumB: eth0 = 10.0.2.30. Estas dos eth0 se conectan a un enrutador cuya dirección privada es 10.0.2.1. Para instalar Heartbeat en Debian ejecutamos `apt-get install heartbeat`

*La información detallada se puede consultar en
file:///usr/share
/doc/heartbeat
/GettingStarted.html.

heartbeat-dev; también podríamos instalar heartbeat-gui, pero no lo utilizaremos en esta guía. Esta instalación creará el directorio `/etc/ha.d` para configurar el clúster de dos nodos de HA y deberá contener tres archivos `ha.cf` (*main configuration file*) `haresources` (*resource configuration file*) y `authkeys` (*authentication information*). El fichero `/etc/ha.d/ha.cf` (se puede obtener un ejemplo desde `/usr/share/doc/heartbeat/`) es el principal fichero de Heartbeat y debería tener las opciones necesarias para que Heartbeat funcione correctamente (es necesario comentar que estas configuraciones son válidas para ambos nodos).

```
# Interfaz de red donde se levantará heartbeat
bcast eth0
# Pings a la interfaz del router para validar que hay conexión a Internet
ping 10.0.2.1
# Pings cada 2 segundos al otro nodo
keepalive 2
# Ausencia de respuesta de 5 segundos hace que este nodo esté en alerta
warntime 5
# Ausencia de respuesta de 10 segundos el otro nodo se considera fuera de servicio
deadtime 10
# Puerto UDP por defecto de Heartbeat según IANA
udpport 694
auto_failback on
# Declaración de los dos nodos
node NteumA
node NteumB
```

Archivo `/etc/ha.d/haresources` donde se identifica el nodo primario como [MasterHost] [IP_Virtual] [servicios]

```
NteumA 10.0.2.15 httpd smb
```

La `IP_Virtual` es la que levantará Heartbeat para referirse al *aliasing* de red, es decir, la que utilizará tanto si `NteumA` (primario) cae como si no, de tal manera que si `NteumA` funciona realmente irá a la dirección `10.0.2.20` y, en caso contrario, a la `10.0.2.30` (`NteumB`). Los servicios disponibles se nombran tal como aparecen en `/etc/init.d`, puesto que es donde Heartbeat los gestionará.

El archivo `/etc/ha.d/authkeys` proporciona una medida de seguridad para evitar que haya suplantación de identidad permitiendo autenticación en sha, md5 o crc (menos seguro). Este archivo debería tener atributos de `600` (`chmod 600 /etc/ha.d/authkeys`) y su contenido será algo como:

```
auth 1
1 sha1 key-for-shal-cualquier-texto-que-se-desea
```

No menos importante es configurar el `/etc/hosts`, por ejemplo:

```
127.0.0.1 localhost
10.0.2.15 router
10.0.2.20 NteumA
10.0.2.30 NteumB
```

Puesto que Heartbeat utiliza el tráfico ICMP para comunicarse entre nodos y en nuestro caso el enrutador, se deberá modificar el cortafuegos para que permita este tipo de tráfico (es importante también considerar que se generarán paquetes ARP y que algún software como Snort u otros IDS pueden considerar esto como ataques de *ARP-spoofing*).

Para probar la funcionalidad, primero asignamos la direcciones manualmente (o por DHCP) a las eth0 de los servidores:

```
ifconfig eth0 down ; ifconfig eth0 up 10.0.2.20 netmask 255.255.255.0 (NteumA)
ifconfig eth0 down ; ifconfig eth0 up 10.0.2.30 netmask 255.255.255.0 (NteumB)
```

Ejecutando `/etc/init.d/heartbeat start` ponemos en marcha Heartbeat en ambos nodos (se recomienda poner en marcha primero el nodo primario y luego, el secundario). Para probar su funcionalidad, presuponemos que ambos nodos están funcionando, pero solo el nodo primario (NteumA) está dando el servicio y el Heartbeat del nodo secundario (NteumB) está enviando continuamente *pings* para comprobar que efectivamente NteumA sigue haciendo su trabajo. Para ello se puede intentar hacer una conexión http a la dirección 10.0.2.15 (la del `haresources`) y la respuesta será que se está en NteumA (primario); si apagamos (desconectamos el cable de red de eth0) el nodo primario y se vuelve a entrar 10.0.2.15 será NteumB quien deberá dar la respuesta (si todo ha funcionado correctamente, si no desplegará un mensaje de *Not Found* o *Forbidden*). Para mayor detalle consultad la documentación en `/usr/share/doc/heartbeat`.

Actividades

1. Realizad una monitorización completa del sistema con las herramientas que consideréis adecuadas y haced un diagnóstico de la utilización de recursos y cuellos de botella que podrían existir en el sistema. Simular la carga en el sistema del código de `sumdis.c` dado en el módulo “Clusterización”. Por ejemplo, utilizad: `sumdis 1 2000000`.
2. Cambiad los parámetros del núcleo y del compilador y ejecutad el código mencionado en la actividad anterior (`sumdis.c`) con, por ejemplo: `time ./sumdis 1 1000000`.
3. Con la ejecución de las dos actividades anteriores extraed conclusiones sobre los resultados.
4. Con el programa de iteraciones indicado en este módulo, implementad un forma de terminar la ejecución en 5 segundos, independiente de las llamadas al sistema excepto que sea solo una vez.
5. Monitorizad todos los programas anteriores con Munin.
6. Registrad cuatro servicios con Monin y haced la gestión y seguimiento por medio del programa.
7. Instalad MRTG y monitorizad la CPU de la ejecución de los programas anteriores.
8. Instalad y experimentad con los sistemas descritos de alta disponibilidad.

Bibliografía

- [1] **Bravo E., D.** (2006). *Mejorando la Performance en Sistemas Linux/Unix*. GbuFDL1.2. <die-gobravoestrada@hotmail.com>
- [2] *Cacti*.
<<http://cacti.net/>>
- [3] *Documentación de Linux-HA*.
<<http://www.linux-ha.org/doc/users-guide/users-guide.html>>
- [4] *Frysk*.
<<http://sources.redhat.com/frysk/>>
- [5] *Howto sobre instalación de MRTG en Debian*.
<<http://preguntaslinux.org/-howto-instalacion-de-mrtg-monitoreo-debian-t-3061.html>>
- [6] *Instalación de Nagios sobre Debian*.
<http://debianclusters.org/index.php/Nagios_Installation_and_Configuration>
<<http://debianclusters.org/index.php/Nagios>>
- [7] **Leung, C. T.** *Building a Two-Node Linux Cluster with Heartbeat*.
<<http://www.linuxjournal.com/article/5862>>
- [8] *Linux-HA*.
<http://linux-ha.org/wiki/Main_Page>
- [9] **Majidimehr, A.** (1996). *Optimizing UNIX for Performance*. Prentice Hall.
- [10] *Monit*.
<<http://mmonit.com/monit/>>
- [11] *Monitor Debian servers with monit*.
<<http://www.debian-administration.org/articles/269>>
- [12] *Monitorización con Munin y monit*.
<http://www.howtoforge.com/server_monitoring_monit_munin>
- [13] *Monitorización con SNMP y MRTG*.
<http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO:_Ch22:_Monitoring_Server_Performance>
- [14] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.

- [15] *MRTG*.
<<http://oss.oetiker.ch/mrtg/>>
- [16] *Munin*.
<<http://munin.projects.linpro.no/>>
- [17] *Optimización de servidores Linux*.
<http://people.redhat.com/alikins/system_tuning.html>
- [18] *Pacemaker*.
<http://clusterlabs.org/wiki/Main_Page>
- [19] *Performance Monitoring Tools for Linux*.
<<http://www.linuxjournal.com/article.php?sid=2396>>