Sintonización, optimización y alta disponibilidad

Remo Suppi Boldrito

PID_00212471

Universitat Oberta de Catalunya

Índice

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 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.

Por otro lado con las necesidades de servicios actuales, los usuarios son muy exigentes con la calidad de servicio que se obtiene y es por ello que un administrador debe prevenir las incidencias que puedan ocurrir en el sistema antes de que las mismas ocurran. Para ello es necesario que el administrador "vigile" de forma continuada determinados parámetros de comportamiento de los sistemas que le puedan ayudar en la toma de decisiones y actuar "en avance" evitando que se produzca el error, ya que si esto pasara podría suponer la posibilidad de que el sistema dejara de funcionar con las posibles consecuencias económicas en algunos casos, pero casi siempre con el deterioro de la imagen del empresa/institución que esto conlleva (a nadie le agradará -o le debería agradar- que los usuarios le informen de un fallo en sus sistemas de información).

En resumen, un sistema de monitorización debe ayudar al administrador a reducir el MTTR (*Mean time to recovery*) que indica la media de tiempo que un sistema tarda en recuperarse de un fallo y que puede tener un valor dentro del contrato de calidad de servicio (normalmente llamado SLA *Service Level Agreement*) por lo cual también puede tener consecuencia económicas. Este valor a veces se le llama *"mean time to replace/repair/recover/resolve"* en función

de qué sistema se trate y qué SLA se haya acordado, pero en todos estamos hablando de la ventana de tiempo entre la cual se detecta un problema y las acciones para solucionarlo. Con la monitorización podremos tener indicios de estos problemas y ejecutar las acciones para que no lleguen a más y que si ocurren, tengan el MTTR más bajo posible.

Ved también

La seguridad se estudia en el módulo "Administración de seguridad".

En este módulo se verán las principales herramientas para monitorizar un sistema GNU/Linux, como son Munin, Monit, MRTG, Ganglia, Nagios, Cactis o Zabbix y se darán indicaciones de cómo sintonizar el sistema a partir de la información obtenida.

Es importante notar que si el MTTR es cercano a cero, el sistema tendrá que tener redundancia en los otros los sistemas y es un aspecto importante en la actualidad para los servidores de sistemas de la información, que se conoce como alta disponibilidad.

La alta disponibilidad (*high availability*) es un protocolo de diseño del sistema y su implementación asociada 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 para LVS -Linux Virtual Server-, OpenSAF, Piranha, UltraMonkey, Pacemaker+Corosync, o Kimberlite (obs:EOL), etc.), en este módulo analizaremos cómo se desarrolla una arquitectura redundante en servicios utilizando Heartbeat+DRBD.

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 de monitorización y su integración para resolver los problemas de eficiencias/disponibilidad.
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 [31]. 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 directorio `/proc` lo veremos como un directorio, pero en realidad es un sistema de archivos ficticio llamado procfs (y que se monta en tiempo de *boot* de la máquina), 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. El `/proc` actúa como interfaz a la estructura de datos internos de núcleo y puede ser utilizado para cambiar cierta información del kernel en tiempo de ejecución (sysctl). No debemos confundir **procfs** con el **sysfs** ya que este último exporta información sobre los dispositivos y sus controladores desde el modelo de dispositivos del núcleo hacia el espacio del usuario (y es utilizado por algunas partes importantes del sistema como el udev que es el que crea por compatibilidad el /dev) permitiendo obtener parámetros y configurar alguno de ellos (p.ej., saber el tamaño de un disco `cat /sys/block/sda/size` o que dispositivos tenemos en /sys/class). Una vista de este directorio es:

```
bus          locks           cgroups         meminfo
cmdline      misc            consoles        modules
cpuinfo      mounts          crypto          mtrr
devices      net             diskstats       pagetypeinfo
dma          partitions      dri             sched_debug
driver       self            execdomains     slabinfo
fb           softirqs        filesystems     stat
fs           swaps           interrupts      sys
iomem        sysrq-trigger   ioports         sysvipc
```

```
irq          timer_list      kallsyms        timer_stats
kcore        tty             keys            uptime
key-users    version         kmsg            vmallocinfo
acpi         kpagecount      vmstat          asound
kpageflags   zoneinfo        buddyinfo       loadavg

Además de una serie de directorios numéricos que corresponde a cada uno
de los procesos del sistema.
```

En el directorio `/proc` existen un conjunto de archivos y directorios con diferente información. 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!

Ved también

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 del 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
I (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 *swap*
r	Memoria libre y ocupación de *swap*
I (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 utilizado 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` $\simeq$ 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` $\simeq$ 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 (en modo gráfico está `gnome-system-monitor`):

```
top - 15:09:08 up 21 min,  2 users,  load average: 0.16, 0.15, 0.12
Tasks: 184 total,   2 running, 182 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  2.8 sy,  0.0 ni, 96.8 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1509992 total,   846560 used,   663432 free,   117304 buffers
KiB Swap:  1087484 total,        0 used,  1087484 free,   374076 cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4144 root      20   0  201m  36m 8448 S   9.6  2.5   0:39.35 Xorg
 4694 adminp    20   0  980m  64m  29m S   6.7  4.4   0:26.22 gnome-shell
 4730 adminp    20   0  363m  16m  10m S   2.3  1.1   0:04.04 gnome-terminal
 4221 root      20   0 69796 1776 1140 S   0.3  0.1   0:01.62 nmbd
 4655 adminp    20   0  571m  26m  13m S   0.3  1.8   0:01.00 gnome-settings-
 6287 root      20   0 15080 1520 1072 R   0.3  0.1   0:00.10 top
    1 root      20   0 10648  812  676 S   0.0  0.1   0:01.21 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:00.93 ksoftirqd/0
```

Un comando interesante y que presenta la información de otra forma que puede servir para tener una panonrámica de todo el sistema es `atop` (se debe instalar `apt-get install atop`). A continuación unas líneas de este comando nos muestran su potencialidad:

```
ATOP - SySDW         2014/06/28  10:55:42         ------          17m58s elapsed
PRC | sys    2m06s | user   9.10s | #proc    184 | #zombie    0 | #exit      0 |
CPU | sys       7% | user      1% | irq       1% | idle    388% | wait      3% |
CPL | avg1    0.03 | avg5    0.07 | avg15   0.10 | csw  1116790 | intr  619733 |
MEM | tot     1.4G | free  659.0M | cache 369.7M | buff  100.2M | slab   64.5M |
SWP | tot     1.0G | free    1.0G |              | vmcom   2.1G | vmlim   1.8G |
DSK |          sda | busy      3% | read   27461 | write   2710 | avio 1.03 ms |
NET | eth0      0% | pcki     278 | pcko     260 | si    2 Kbps | so    0 Kbps |
NET | lo      ---- | pcki      12 | pcko      12 | si    0 Kbps | so    0 Kbps |
              *** system and process activity since boot ***
  PID  SYSCPU  USRCPU  VGROW  RGROW  RDDSK   WRDSK ST EXC S  CPU  CMD       1/27
 3865  67.00s   2.31s 199.6M 36676K 14196K    148K N-   - S   7%  Xorg
 4505  40.21s   4.14s 980.6M 68848K 25396K      8K N-   - R   4%  gnome-shell
 4543   4.14s   0.60s 427.4M 16048K  4756K    160K N-   - S   0%  gnome-terminal
```

También se pueden utilizar las herramientas del paquete `systat` 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).

Un resumen de los comandos más interesantes es:

Comando	Descripción
atop, top	Actividad de los preocesos
arpwatch	monitor de Ethernet/FDDI
bmon, bwm-ng, nload	monitor del ancho de banda
downtimed	Monitor del tiempo de caída, fuera de servicio
free	Utilización de memoria
iostat, iotop	Actividad de disco y E/S
ip monitor, rtmon, iptotal, iptraf	Monitor de dispositivos de red
mpstat	Estadísticas del procesador
netstat	Estadística de la red
nfswatch	Monitor de NFS
ps, pstree, god	Muestra estado y características de los procesos
/proc	Sistema de archivos virtual
sar, atsar	Recoge información del sistema
stat, iwatch	Estadísticas del sistema de archivos
strace	Eventos de las llamadas al sistemas y señales
tcpdump, etherape, sniffit	Volcado/monitor de paquetes de red
uptime, w	Carga media del sistema y tiempo desde el inicio
vmstat	Estadísiticas del uso de memoria
gnome-system-monitor, gkrellm, xosview, xwatch	Monitores gráficos del sistema
xconsole	Monitor de mensajes en el escritorio

También existen una serie de programas que miden las prestaciones del sistema (*benchmark*) o parte de él como por ejemplo: netperf, mbw (red y ancho de banda), iozone (E/S), sysbench, globs, gtkperf, hpcc (general), bonie++ (disco). Es importante destacar que el benchmark hpcc incluye el *High-Performance* LINPACK (HPL) *benchmark* que es el utilizado para medir las prestaciones y realizar el *ranking* de las máquinas más potentes del mundo*.

*http://www.top500.org/

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 obtener 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 conflictiva 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 `systune` y `systunedump`, que permiten descargar las variables en un archivo y modificarlas, para cargarlas nuevamente en el `/proc` (el comando `systune` guarda la configuración en `/etc/systune.conf`). En este caso, por ejemplo, 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).

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

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-x.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` con el directorio actual (indicado por `./`), puede tener una relación desfavorable de ejecución (esto es debido a que la llamada `exec()` no guarda nada en caché y perjudica esta 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 [31].

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).

Enlaces de interés

Es interesante consultar los siguientes libro/artículos:
www.redbooks.ibm.com sobre Linux Performance and Tuning Guidelines, http://people.redhat.com/alikins/system_tuning.html sobre información de optimización de sistemas servidores Linux y http://www.linuxjournal.com/article/2396 sobre *Performance Monitoring Tools for Linux*. El primero es un e-book abierto de la serie RedBooks de IBM muy bien organizado y con gran cantidad de detalles sobre la sintonización de sistemas Linux, los dos restantes son artículos que si bien tienen un cierto tiempo, los conceptos/metodología y algunos procedimientos continúan vigentes.

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:

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).

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`.

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 ([4] 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`

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 [4]:

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 clomuna1=# es un comentario.
# El orden de la líneas es importante
# Formato: permission : users : origins
# Deshabilar 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 solo 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 8 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). Utilizar algoritmos como SHA512 para la configuración de paswssd (en Debian viene configurado por defecto, ver */etc/pam.d/common-password*).

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 `shh` 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`. Verificar que los archivos de configuración del *bootloader* (`/etc/ lilo.conf` si tenemos Lilo como gestor de arranque, o /etc/grub.d si trabajamos con Grub2) debe tener los permisos adecuados para que nadie lo pueda modificar excepto el root. Mediante los archivos de configuración de *boot* se permiten una serie de opciones que es conveniente considerar: `timeout` para controlar el tiempo de *boot*; `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 (consultar los manuales de Lilo/Grub para la sintaxis correcta).

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 recuerde 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 al final de la línea borramos el parámetro `ro` e introducimos `rw init=/bin/bash` (lo cual indica acceso directo a la consola). Presionamos "F10"

d) Con esto se arrancará 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 apague 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/gsahdow` 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 -prem -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` (`sar` 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 LibreOffice, 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 [3]:

```
#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 las 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`. [3]

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, Nagios, Ganglia, Zabbix y Cacti.

1.2.1. Munin

Munin [8] 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 sistema se llama `sysdw.nteum.org` y que ya la tenemos configurada con este nombre y con el DocumentRoot de Apache 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
```

Por último munin solo viene configurado para conectarse desde el localhost, si lo deseamos hacer desde otra máquina debemos cambiar entonces el archivo */etc/apache2/conf.d/munin* (que es un link a */etc/munin/apache.conf*) comentando la línea `#Allow from localhost 127.0.0.0/8 ::1` por `Allow from all` y reiniciar Apache (`service apache2 restart`).

Después de unos minutos se podrán ver los primeros resultados en la dirección web http://localhost/munin en el navegador (o también el dominio que tene-

mos asignado en */etc/hosts* por ejemplo en nuestro caso *sysdw.nteum.org*). 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 /etc/munin/htpasswd
   require valid-user
```

Después se debe crear el archivo de contraseña en `/etc/munin/htpasswd` con el comando (como root): `htpasswd -c /etc/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.

Munin viene con un conjunto de *plugins* instalados pero fácilmente se pueden habilitar otros haciendo, por ejemplo para monitorizar MySQL:

```
cd /etc/munin/plugins
ln -s /usr/share/munin/plugins/mysql_ mysql_
ln -s /usr/share/munin/plugins/mysql_bytes mysql_bytes
ln -s /usr/share/munin/plugins/mysql_innodb mysql_innodb
ln -s /usr/share/munin/plugins/mysql_isam_space_ mysql_isam_space_
ln -s /usr/share/munin/plugins/mysql_queries mysql_queries
ln -s /usr/share/munin/plugins/mysql_slowqueries mysql_slowqueries
ln -s /usr/share/munin/plugins/mysql_threads mysql_threads
```

1.2.2. Monit

Monit [7] 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` [32]:

*http://mmonit.com/monit

```
# Monit control file example: /etc/monit/monitrc
# Solo se muestran las líneas cambiadas
    set daemon 120 # Poll at 2-minute intervals
    set logfile /var/log/monit.log
    set alert adminp@sysdw.nteum.org
# Se utiliza el servidor interno que dispone monit para controlar apache2 también
set httpd port 2812 and
    use address localhost  # only accept connection from localhost
    allow admin:monit      # require user 'admin' with password 'monit'
# Ejemplos de Monitores
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 process apache with pidfile /var/run/apache2.pid
   group www-data
   start program = "/etc/init.d/apache2 start"
   stop program  = "/etc/init.d/apache2 stop"
   if failed host sysdw.nteum.org port 80 protocol http
      and request "/monit/token" then restart
   if cpu is greater than 60% for 2 cycles then alert
   if cpu > 80% for 5 cycles then restart
check process ntpd with pidfile /var/run/ntpd.pid
   start program = "/etc/init.d/ntp start"
   stop  program = "/etc/init.d/ntp stop"
   if failed host 127.0.0.1 port 123 type udp then restart
   if 5 restarts within 5 cycles then timeout
```

Enlace de interés

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

Para la monitorización de Apache hemos indicado que verifique un fichero que deberás estar en */var/www/monit/token* por lo cual se deberá crear con: `mkdir /var/www/monit; echo "hello" >/var/www/monit/token`. 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 `service monit restart` (verificar el valor de la variable START en `/etc/default/monit`.

1.2.3. SNMP + MRTG

El **MRTG** (*Multi-Router Traffic Grapher*) [9] 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. En primer lugar instalaremos SNMP o Protocolo Simple de Administración de Red (*Simple Network Management Protocol*) que es un protocolo de la capa de aplicación que facilita la obtención información entre dispositivos de red (p. ej., *routers*, *switches*, servidores, estaciones de trabajo, impresoras, ...) y que permite a los administradores supervisar el funcionamiento de la red y buscar/resolver sus problemas. Para ello hacemos `apt-get install snmp snmpd`. Las variables accesibles a través de SNMP están organizadas en jerarquías metadatos (tipo, descripción, ...) y están almacenadas en una tablas llamadas *Management Information Bases* (MIBs). Para instalarlas debemos agregar primero al repositorio de Debian en non-free y luego descargar el paquete:

Agregamos en /etc/apt/sources.list
deb http://ftp.debian.org/debian wheezy main contrib non-free

Luego actualizamos los repositorios e instalamos el paquete

apt-get update
apt-get install snmp-mibs-downloader
download-mibs

Luego debemos configurar el servicio `snmpd` y para ello editamos la configuración */etc/snmp/snmpd.conf* -solo hemos dejado las líneas más importantes a cambiar-:

agentAddress udp:127.0.0.1:161
rocomunity public
com2sec local localhost public
group MyRWGroup v1 local
group MyRWGroup v2c local
group MyRWGroup usm local
view all included .1 80
access MyRWGroup "" any noauth exact all all none
com2sec notConfigUser default mrtg
group notConfigGroup v1 notConfigUser
group notConfigGroup v2c notConfigUser
view systemview included .1.3.6.1.2.1.1
view systemview included .1.3.6.1.2.1.25.1.1
view systemview included .1 80
access notConfigGroup "" any noauth exact systemview none none
syslocation BCN
syscontact Adminp <adminp@sysdw.nteum.org>

A continuación en el archivo */etc/defaults/snmpd* hemos modificado la línea export `MIBS=/usr/share/mibs` para indicarle donde estaban las MIBs y se reinicia el servicio (`/etc/init.d/snmpd restart`). Podemos interrogar al servidor `snmpd` probar utilizando el comando `snmpwalk`, por ejemplo:

`snmpwalk -v1 -c public localhost` Dará una larga lista de información
`snmpwalk -v 2c -c public localhost` Idem anterior
O preguntarle por una variable específica de la MIB:
`snmpwalk -v1 -c mrtg localhost IP-MIB::ipAdEntIfIndex`
IP-MIB::ipAdEntIfIndex.127.0.0.1 = INTEGER: 1
IP-MIB::ipAdEntIfIndex.158.109.65.67 = INTEGER: 2

Con esto ya podemos instalar MRTG haciendo

```
apt-get install mrtg mrtg-contrib mrtgutils
```

Luego generamos la configuración con

```
cfgmaker public@localhost >/etc/mrtg.cfg
```

y debemos crear el directorio y cambiar las protecciones para el grupo de Apache: `/var/www/mrtg; chown www-data:www.data /var/www/mrtg`. Por último deberemos crear el índex.html con:

```
indexmaker --title="Localhost" --output /var/www/mrtg/index.html /etc/mrtg.cfg.
```

Antes de ejecutar mrtg existe un error en Debian Wheezy y IPv6 que podemos corregir con [10]:

> Cuando ejecutamos LANG=C /usr/bin/mrtg /etc/mrtg.cfg obtenemos el error:
> Subroutine SNMP_Session::pack_sockaddr_in6 redefined ...
> Solución: editar el archivo /usr/share/perl5/SNMP_Session.pm
> En la línea 149:
> Donde dice: import Socket6;
> Reemplazar por: Socket6->import(qw(inet_pton getaddrinfo));
> En la línea 609:
> donde dice: import Socket6;
> Reemplazar por: Socket6->import(qw(inet_pton getaddrinfo));

Para ejecutar `mrtg` inicialmente y verificar si todo está bien deberíamos hacer: `env LANG=C /usr/bin/mrtg /etc/mrtg.cfg` (y repitiéndola una serie de veces) podremos visualizar la actividad de red accediendo a la URL: *http://sysdw.nteum.org/mrtg/* y podremos ver que comienza a mostrar la actividad de eth0.

Finalmente si está todo bien deberemos configurar el `cron` para que se actualizacen las gráficas cada 5', para ello editamos el `crontab -e` insertando `*/5 * * * * root env LANG=C /usr/bin/mrtg /etc/mrtg.cfg`.

MRTG es muy potente para visualizar gráficos de variables ya sea SNMP o de scripts que podemos incluir. Veamos dos ejemplos donde el primero lo utilizaremos para medir la carga de CPU basado en una consulta SNMP al servidor y la segunda ejecutaremos un script para ver los procesos y los de root del sistema (con diferentes opciones de colores por ejemplo).

```
Para la carga de CPU agregar al final de /etc/mrtg:
    LoadMIBs: /usr/share/mibs/netsnmp/UCD-SNMP-MIB
    Target[localhost.cpu]:ssCpuRawUser.0&ssCpuRawUser.0:public@127.0.0.1+
    ssCpuRawSystem.0&ssCpuRawSystem.0:public@127.0.0.1+
    ssCpuRawNice.0&ssCpuRawNice.0:public@127.0.0.1
    RouterUptime[localhost.cpu]: public@127.0.0.1
    MaxBytes[localhost.cpu]: 100
    Title[localhost.cpu]: CPU Load
    PageTop[localhost.cpu]: <H1>CPU Load %</H1>
    Unscaled[localhost.cpu]: ymwd
    ShortLegend[localhost.cpu]: %
    YLegend[localhost.cpu]: CPU Utilization
    Legend1[localhost.cpu]: Active CPU in % (Load)
    Legend2[localhost.cpu]:
    Legend3[localhost.cpu]:
    Legend4[localhost.cpu]:
    LegendI[localhost.cpu]: Active
    LegendO[localhost.cpu]:
    Options[localhost.cpu]: growright,nopercent


Para el número de procesos agregar al final de /etc/mrtg:
    Title[procesos]: Processes
    Target[procesos]:`/usr/local/bin/proc.sh`
    PageTop[procesos]: <h1>Processes</h1>
    MaxBytes[procesos]: 200
    YLegend[procesos]: Processes
    ShortLegend[procesos]: Num.
    XSize[procesos]: 300
```

```
YSize[procesos]: 100
Options[procesos]: nopercent,gauge
Colours[procesos]: ORANGE\#FF7500,BLUE\#1000ff,DARK GREEN\#006600,VIOLET\#ff00ff
LegendI[procesos]: Processes Root
LegendO[procesos]: Total of Processes
```

Donde el script proc.sh es:

```
#!/bin/bash
sysname=`hostname`
p1=`ps -edaf | wc -l`
p1=`expr  $p1 - 1`\\
p2=`ps -edaf  | grep ^root \| wc -l`
p2=`expr $p2 - 2`

echo $p2
echo $p1
echo $sysname
```

Se debe tener en cuenta de ejecutar el

```
indexmaker --title="Localhost" --output /var/www/mrtg/index.html /etc/mrtg.cfg
```

y luego para verificar que todo esta bien `env LANG=C /usr/bin/mrtg /etc/mrtg.cfg`, recargando la URL http://sysdw.nteum.org/mrtg/ veremos las dos nuevas gráficas y su actividad.

En las siguientes referencias [34, 10, 11, 17] se puede encontrar más información sobre SNMP (configuración, solución de errores, etc).

1.2.4. Nagios

Nagios es un sistema de monitorización de redes y sistemas ampliamente utilizado por su versatilidad y flexibilidad sobre todo a la hora de comunicar alertas de comportamientos o tendencias de los sistemas monitorizados. Puede monitorizar servicios de red (SMTP, HTTP, SNMP...), recursos del sistema (carga del procesador, uso de los discos, memoria, estado de los puertos...) tanto locales como remotos a través de un *plugin* (llamado NRPE) y se puede extender su funcionalidad a través de estos *plugins*. El producto base de Nagios es llamado **Nagios Core** el cual es *Open Source* sirve de base a otros productos comerciales de Nagios como NagiosXI, IM, NA. Nagios tiene una comunidad muy activa (llamada Nagios Exchange) y en la página web de Nagios (http://www.nagios.org/) se puede acceder a las contribuciones, *plugins* y documentación. Nagios permite consultar prácticamente cualquier parámetro de interés de un sistema, y genera alertas, que pueden ser recibidas por los responsables correspondientes mediante diferentes canales como por ejemplo correo electrónico y mensajes SMS. La instalación básica es muy simple haciendo `apt-get install nagios3` que instalara todas las librerías y plugins para una primera configuración. Durante la

instalación se nos solicitará un *passwd* pero que también posteriormente se puede cambiar haciendo: `cd /etc/nagios3; htpasswd htpasswd.users nagiosadmin`. Luego podremos recargar nuevamente Apache2 y conectando a la URL *http://sysdw.nteum.org/nagios3/* se solicitará el acceso como *nagiosadmin* y el *passwd* que le hemos introducido y podremos ver la estructura de Nagios y observar los servicios monitorizados en cada uno de los apartados.

El archivo de configuración de nagios está en */etc/nagios3/nagios.cfg* y el cual incluye todos los archivos del directorio conf.d del mismo directorio. En ellos tendremos agrupados por archivos los diferentes aspectos a monitorizar, por ejemplo sistemas (localhost_nagios2.cfg) y servicios (services_nagios2.cfg). El fichero más importante probablemente es *localhost_nagios2.cfg* del cual haremos una breve descripción:

```
# definición de un host utilizando un template (generic-host)
define host{
        use                     generic-host
        host_name               localhost
        alias                   localhost
        address                 127.0.0.1
        }
# definición de un servicio -espacio de disco- utilizando un template
# (generic-service) ejecutando un cmd y con advertencia al 20% y error al 10%
# del espacio libre.
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Disk Space
        check_command                   check_all_disks!20%!10%
        }
# ídem para usuarios: advertencia 20 usuarios, crítico 50 usuario
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Current Users
        check_command                   check_users!20!50
        }
#ídem procesos:advertencia 250, crítico 400
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Total Processes
check_command                   check_procs!250!400
        }
# Carga de CPU.
define service{
        use                             generic-service
        host_name                       localhost
        service_description             Current Load
check_command                   check_load!5.0!4.0!3.0!10.0!6.0!4.0
        }
```

Si quisiéramos agregar un servicio (por ejemplo ping al localhost) solo deberíamos agregar al final del archivo:

```
# Ping: advertencia cuando el 20% sea 100ms, crítico 60% sea 500ms.
define service{
        use                             generic-service
        host                            localhost
```

```
service_description             PING
check_command                   check_ping!100.0,20%!500.0,60%
}
```

Los *plugins* son incorporados por */etc/nagios3/nagios.cfg* y están definidos en */etc/nagios-plugins/config* donde se pueden ver las diferentes alternativas para monitorizar.

Dos complementos interesantes para Nagios son PNP4Nagios* y NagVis:

*http://docs.pnp4nagios.org/pnp-0.6/start

1) PNP4Nagios permite almacenar las información que proveen los *plugins* y almacenarla en una base de datos llamada RRD-database y llamar a la herramienta RRD Tool (http://oss.oetiker.ch/rrdtool/) que permite la visualización de estos datos incrustados en la página de Nagios. Por lo cual además del valor instantáneo de las variables a monitorizar también tendremos integrados estos en el tiempo y podremos ver su evolución.

2) NavVis (http://www.nagvis.org/) permite dibujar (sección map) la red de diferentes formas y aspectos para tener diferentes visualizaciones de la red monitorizada en forma activa, es decir viendo los valores de las variables seleccionadas sobre el gráfico.

Por último es interesante considerar **Icinga** (https://www.icinga.org/) que es una bifurcación (*fork*) de Nagios (del 2009) y ha evolucionado mucho en el último tiempo (para algunos ya ha superado a Nagios) y su objetivo es transformarse en una herramienta de referencia dentro del *Open Source* y en el ámbito de la monitorización de redes y sistemas. Icinga nace con la idea de superar las deficiencias en el proceso de desarrollo de Nagios y sus políticas, así como la voluntad de ser más dinámica y fácil agregar nuevas características, como por ejemplo una interfaz de usuario de estilo Web2.0, conectores de base de datos adicionales y una API REST que permita a los administradores integrar numerosas extensiones sin complicadas modificaciones del núcleo. Icinga está disponible en Debian y su configuración es prácticamente similar a Nagios.

1.2.5. Ganglia

Ganglia [12] es una herramienta que permite monitorizar de forma escalable y distribuida el estado de un conjunto de máquinas agrupadas bajo diferentes criterios (red, servicios, etc) o simplemente bajo una misma identificación que llamaremos clúster. La aplicación muestra al usuario las estadísticas de forma remota (por ejemplo, los promedios de carga de la CPU o la utilización de la red) de todas las máquinas que conforman este el clúster basándose en un diseño jerárquico y utiliza comunicaciones punto-a-punto o *multicast* para el intercambio de información entre los diferentes nodos que forman el clúster.

Ganglia utiliza XML para la representación de datos, XDR para el transporte compacto y portátil de datos y RRDtool (http://oss.oetiker.ch/rrdtool/) para almacenamiento de datos y visualización. El sistema se compone de dos *daemons* (gmond y gmetad), una página de visualización (ganglia-webfrontend) basada en PHP.

Gmond es un *daemon* multihilo que se ejecuta en cada nodo del clúster que se desea supervisar (no es necesario tener un sistema de archivos NFS o base de datos ni mantener cuentas especiales de los archivos de configuración). Las tareas de Gmond son: monitorizar los cambios de estado en el *host*, enviar los cambios pertinentes, escuchar el estado de otros nodos (a través de un canal *unicast* o *multicast*) y responder a las peticiones de un XML del estado del clúster. La federación de los nodos se realiza con un árbol de conexiones punto a punto entre los nodos determinados (representativos) del clúster para agregar el estado de los restantes nodos. En cada nodo del árbol se ejecuta el daemon **Gmetad** que periódicamente solicita los datos de los restantes nodos, analiza el XML, guarda todos los parámetros numéricos y exporta el XML agregado por un socket TCP. Las fuentes de datos pueden ser *daemons* gmond, en representación de determinados grupos, u otros *daemons* gmetad, en representación de conjuntos de grupos. Finalmente la web de Ganglia proporciona una vista de la información recogida de los nodos del clúster en tiempo real. Por ejemplo, se puede ver la utilización de la CPU durante la última hora, día, semana, mes o año y muestra gráficos similares para el uso de memoria, uso de disco, estadísticas de la red, número de procesos en ejecución y todos los demás indicadores de Ganglia.

Para la instalación de Ganglia sobre Debian (es similar para otras distribuciones):

1) Instalar los paquetes de Ganglia sobre el servidor web: `apt-get install ganglia-monitor gmetad ganglia-webfrontend`

2) Sobre todos los otros nodos solo se necesita tener instalado el paquete `ganglia-monitor`.

3) Los archivos de configuración están en */etc/ganglia* y en gmond.conf la línea más importante es `data_source "my cluster"localhost` donde indica cual será el nombre del cluster (para agregar todas las máquinas bajo el mismo tag y donde se recogerán los datos (en este caso en localhost). En gmond.conf tenemos las configuraciones generales y de nombres además de los canales de comunicación que por defecto son *multicast*. Si deseamos que sean *unicast* deberemos modificar las secciones udp_send|recv donde la IP de host será la del servidor (gmetad) y comentar las direcciones de *multicast*:

```
udp_send_channel {
#  mcast_join = 239.2.11.71
 host = IP_nodo_gmetad
  port = 8649
  ttl = 1
```

```
}

udp_recv_channel {
#  mcast_join = 239.2.11.71
  port = 8649
#  bind = 239.2.11.71
}
```

4) Por último debemos crear el link entre la configuración de Ganglia-frontend y Apache haciendo

```
ln -s /etc/ganglia-webfrontend/apache.conf /etc/apache2/conf.d/ganglia
```

Reiniciamos Apache (`service apache2 restart`) y nos conectamos a URL *http://sysdw.nteum.org/ganglia/* para visualizar una panorámica global de los sistemas monitorizados y haciendo click en cada imagen/botón podremos obtener más información sobre cada unos de los recurso monitorizados.

1.2.6. Otras herramientas

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

- **Zabbix** [35] es un sistema de monitorización (*OpenSource*) que permite recoger el estado de diferentes servicios de red, servidores y hardware de red. Este programa utiliza una base de datos (MySQL, PostgreSQL, SQLite, etc) para mejorar las prestaciones y permite la instalación de agentes Zabbix sobre diferentes máquinas a monitorizar aspectos internos como por ejemplo carga de CPU, utilización de red, espacio en disco, etc. También es posible realizar esta monitorización a través de diferentes protocolos como SNMP, TCP y ICMP, IPMI, etc. y soporta una variedad de mecanismos de notificación en tiempo real, incluyendo XMPP. Su instalación no está disponible en los paquetes de Debian (por diferencias en las políticas de Debian y Zabbix desde 2012) pero se pueden obtener el paquete (.deb) desde la web de Zabbix y seguir la guía de instalación disponible en su web*.

***https://www.zabbix.com /documentation/2.0 /manual/installation /install_from_packages**

- **Cacti** [16] 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.

Ved también

Cacti se describe en el módulo "Clúster, Cloud y Devops".

- **Frysk** [20] donde el objetivo del proyecto es crear un sistema de monitorización distribuido e inteligente para monitorizar procesos e hilos.

Existen un conjunto adicional de herramientas no menos interesantes (no se incluyen las ya mencionadas) que incorpora GNU/Linux para la monitorización de diferentes aspectos del sistema (se recomienda ver el `man` de cada herramienta para mayor información):

- **isag**: *Interactive System Activity Grapher*, para la auditoría de recursos hw/sw.
- **mon**: 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.
- **ksensors (lm-sensors)**: monitor de la placa base (temperatura, alimentación, ventiladores, etc.).
- **systune**: herramienta para retirar capacidades asignadas al núcleo en el fichero `/proc/sys/kernel`.
- **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.

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 dependientes 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), o OpenAIS+Corosync+Pacemaker.

El proyecto Linux-HA (Linux de alta disponibilidad) [18] 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|Corosync + Pacemaker [25].

1.3.1. Guía breve de instalación de Heartbeat y Pacemaker (Debian)

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

En este subapartado se dará una breve descripción de cómo construir un clúster de alta disponibilidad de dos nodos con Heartbeat* para por ejemplo disponer de un servidor Apache de alta disponibilidad. Es interesante (aunque un poco antiguo) el artículo [30] y la documentación del sitio web de Linux-HA [19]. 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 VteumB (secundario), con una interfaz a red que la utilizaremos para conectarse a la red tanto entre ellos como desde afuera. Para hacer esta prueba de concepto hemos utilizados dos máquinas virtuales (con la red configurada en modo *bridged* para que se vean en la red) pero que es la misma prueba con dos máquinas físicas. Nuestras máquinas están configuradas como NteumA: eth0 = 192.168.1.201 y VteumB: eth0 = 192.168.1.202, las dos tienen como netmask 255.255.255.0 y gateway 192.168.1.1. Y definimos además una dirección virtual donde prestaremos los servicios por ejemplo Apache 192.168.1.200. Configuramos las máquinas para que se vean entre ellas (a través de un ping) y solo es necesario que tengan una instalación mínima con `apache` y `sshd` y que estén configuradas tanto en nombre (`hostname`) como en */etc/hosts* con `IP FDQN alias` (por ejemplo que para cada máquina haya una línea como `192.168.1.201 nteuma.nteum.org nteuma` para todas las máquinas del clúster y deberemos ver el nombre de la máquina como la hemos configurado en */etc/hosts* en el nombre corto, por ejemplo con `uname -n`). Para instalar y configurar Apache2 + Heartbeat hacemos:

1) `apt-get install apache2`

2) modificamos la configuración de Apache agregando a *etc/apache2/ports.conf* la línea `NameVirtualHost 192.168.1.200:80` (dejando las restantes como están).

3) Para que Apache no esté arrancado desde el inicio (ya que queremos que lo haga Heartbeat) lo quitamos de los scripts rc*.d con `update-rc.d apache2 remove`

4) instalamos el paquete `chkconfig` que luego lo necesitará Heartbeat `apt-get install chkconfig`

5) Instalamos el paquete Heartbeat con `apt-get install heartbeat`. Todas estas acciones anteriores las realizamos en las dos máquinas.

6) sobre nteuma (primario) realizamos la configuración de heartbeat. Los ficheros de configuración están en */etc/ha.d/* y las plantillas para configurarlos están */usr/share/doc/heartbeat/*. Para el archivo ha.cf lo modificamos con lo siguiente:

```
logfile /var/log/cluster.log
logfacility local0
warntime 5
deadtime 30
initdead 120
keepalive 2
bcast eth0
udpport 694
auto_failback on
node nteuma
node vteumb
```

Donde *logfile* y *logfacility*: indican donde estará el archivo de log y el nivel de mensajes que queremos, *warntime* es el tiempo que transcurrirá antes que heartbeat nos avise, *deadtime* el tiempo tras el cual heartbeat confirmará que un nodo ha caído, *initdead* el tiempo máximo que heartbeat esperará a que un nodo arranque, *keepalive* el intervalo de tiempo para comprobar la disponibilidad. Además bcast la forma de comunicación (*broadcast*) y la interfaz y *node* los nodos que forma nuestro cluster HA.

7) El archivo authkeys es donde se configura la comunicación entre los nodos del clúster que deberá tener permisos solo de root (`chmod 600 /etc/ha.d/authkeys`):

```
auth 2
2 sha1 MyPaSsWoRd
```

8) el archivo */etc/ha.d/haresources* le indicamos el nodo primario y el servicio (apache2) a levantar:

```
nteuma IPaddr2::192.168.2.100/24/eth0 apache2
```

9) Ahora debemos propagar la configuración a los nodos del cluster (vteumb en nuestro caso pero se haría a todos los nodos indicados en ha.cf):

`/usr/share/heartbeat/ha_propagate`

10) Reiniciamos las máquinas (reboot) para asegurarnos que todo se inicia como deseamos.

11) Desde el *host* u otra máquina externa que vea las anteriores

Para comprobar cual es el servidor que está prestando el servicio hemos incluido en */var/www/index.html* el nombre de la máquina así cuando carguemos la página a través de la conexión a la IP 192.168.1.200 veremos cual máquina es la que presta el servicio. En primer lugar deberemos ver que es NteumA y si quitamos la red de esta (`ifdown eth0`) veremos que en unos segundos al refrescar la página será VteumB.

Consultar la documentación en la siguiente página web: http://www.linux-ha.org/doc/man-pages/man-pages.html. Como comandos para consultar y ver el estado de nuestro cluster podemos utilizar: `cl_status` para ver el estado de todo el sistema, `hb_addnode` envía un mensaje al cluster para añadir nuevos nodos o `hb_delnode` para quitarlos.

Se puede agregar en *ha.cf* la order `crm respawn` (que es el Cluster Resource Manager -crm- de LinuxHA) que iniciará el *daemon* crmd que nos brindará información sobre el estado del cluster y nos permitirá gestionar con una serie de comandos (crm_*) los recursos del cluster. Por ejemplo `crm_mon -1` nos dará información del estado de nuestro cluster.

Como podremos comprobar si agregamos esta línea nuestro simple cluster dejará de funcionar ya que el control de recurso ahora lo está haciendo Pacemaker (crm) y deberemos configurarlo para tal fin (o quitar la línea crm respawn y reiniciar los servicios). Para mayor información consultar [21, 22, 23, 24].

Para configurar el CRM junto con Hearbeat para un servicio activo-pasivo donde si un nodo falla el nodo pasivo adoptará su función (en nuestro caso Apache). En nuestro caso utilizaremos Heartbeat para mantener la comunicación entre los nodos y Pacemaker como Resource Manager que provee el control y administración de los recursos provistos por el clúster. Básicamente Pacemaker puede manejar diferentes tipos de recursos llamados LSB que serán los que provee GNU/Linux y que pueden gestionarse a través de */etc/init.d* y OCF que nos permitirá inicializar una dirección virtual, monitorizar un recurso, iniciar/parar un recurso, cambiar su orden de servicio, etc.

1) Partimos que ya hemos instalado heartbeat and pacemaker (este último se instala cuando se instala heartbeat) y sino podemos hacer `apt-get install heartbeat pacemaker`

2) A la configuración de */etc/ha.d/ha.cf* mencionada anteriormente agregamos `crm respawn` para iniciar pacemaker (veremos que hay un proceso lammado `crmd`)

3) */etc/ha.d/authkeys* lo dejamos como ya lo teníamos configurado.

4) Reiniciamos el servicio `service heartbeat restart`

5) Ahora podremos ver es estado del cluster con `crm status`

```
============
Last updated: Tue Jul 1 12:06:36 2014
Stack: Heartbeat
Current DC: vteumb (...) - partition with quorum
...
2 Nodes configured, unknown expected votes
0 Resources configured.
============
Online: [ veteumb nteuma ]
```

6) Deshabilitamos stonith que no lo necesitaremos: `crm configure property stonith-enabled=false`

7) Inicializamos los nodos para el quorum:

```
crm configure property expected-quorum-votes="2"
```

8) Para tener quorum, la mitad de los nodos del cluster debe ser online (Nro. de nodos/2)+1 pero en un cluster de 2 nodos esto no ocurre cuando falla un nodo por lo tanto necesitamos inicializar la política a *ignore*: `crm configure property no-quorum-policy=ignore`

9) Para prever el *failback* de un recurso: `crm configure rsc_defaults resource-stickiness=100`

10) Podemos listar los objectos OCF con `crm ra list ocf` y en nuestro caso utilizaremos IPaddr2. Para obteber más información sobre él: `crm ra info ocf:IPaddr2`

11) Agregamos una IP virtual (VIP) a nuestro cluster:

```
crm configure primitive havip1 ocf:IPaddr2 params ip=192.168.1.200
cidr_netmask=32 nic=eth0 op monitor interval=30s
```

12) Verificamos el recurso havip1 se encuentra en el primer nodo ejecutando `crm status` y nos dará una info similar a la anterior pero con una línea como *havip1 (ocf::heartbeat:IPaddr2): Started nteuma*

13) Agregamos el daemon a nuestro cluster: `crm ra info ocf:anything`

14) Agregamos apache a nuestro cluster

```
crm configure primitive apacheha lsb::apache2 op monitor interval=15s
```

15) Inicializamos el recurso VIP y apache en el mismo nodo: `crm configure colocation apacheha-havip1 INFINITY: havip1 apacheha` y veremos con `crm status` que *havip1 (ocf::heartbeat:IPaddr2): Started nteuma apacheha (lsb:apache2): Started nteuma*

16) Podemos configurar el orden de los servicios:

```
crm configure order ip-apache mandatory: havip1 apacheha
```

17) O migrar un servicio a otro nodo: `crm resource migrate apacheha vteumb` (con lo que si nos conectamos a 192.168.1.200 veremos que el servicio es provisto por un servidor o por otro). Este comando lo podemos usar para hacer mantenimientos transfiriendo el servicio de un sitio a otro sin interrumpirlo. Si ejecutamos la instrucción `crm status` veremos entonces *havip1 (ocf::heartbeat:IPaddr2): Started nteuma apacheha (lsb:apache2): Started vteumb*

18) La configuración de nuestro cluster la podemos ver con `crm configure show`:

19) Es interesante ejecutar el navegador con la URL 192.168.1.200 e ir desactivando los nodos o recuperándolos para ver como adopta el rol cada uno y mantienen el servicio (mirar el estado entre cambio y cambio con `crm status`)

Si comentamos la línea `crm respawn` en */etc/ha.d/ha.cf* volveremos a la gestión básica anterior sin el gestor de recursos Pacemaker.

1.3.2. DRBD

El **Distributed Replicated Block Device** (DRBD) es un almacenamiento distribuido sobre múltiples *hosts* donde, a igual que en RAID1, los datos son replicados sobre el sistema de archivo sobre los otros *hosts* y sobre TCP/IP.[26] En esta prueba de concepto utilizaremos las mismas máquinas utilizadas en Heartbeat a las cuales le hemos adicionado una unidad de disco más (*/dev/sdb*) y hemos creado una partición sobre ellas */dev/sdb1*). La instalación es:

1) Instalar en ambas máquinas DRBD: `apt-get install drbd8-utils` donde los archivos de configuración estarán en */etc/drbd.d/* (existe un archivo */etc/drbd.conf* pero que incluye a los archivos del directorio mencionado).

2) Crearemos la configuración del recurso como */etc/drbd.d/demo.res* con el siguiente contenido:

```
resource drbddemo {
   meta-disk internal;
   device /dev/drbd1;
   syncer {
       verify-alg sha1;
   }
   net {
       allow-two-primaries;
   }
   on nteuma {
       disk /dev/sdb1;
       address 192.168.1.201:7789;
   }
   on vteumb {
       disk /dev/sdb1;
       address 192.168.1.202:7789;
   }
}
```

3) Copiamos el archivo:

```
scp /etc/drbd.d/demo.res vteum:/etc/drbd.d/demo.res
```

4) Inicializamos en los dos nodos el dispositivo ejecutando:

```
drbdadm create-md drbddemo
```

5) Sobre nteum (verficar con `uname -n`) hacemos: `modprobe drbd` para cargar el módulo de kernel, luego `drbdadm up drbddemo` para levantar el dispositivo y lo podremos mirar con `cat /proc/drbd` que no indicará (entre otros mensajes) que está:

cs:WFConnection ro:Secondary/Unknown ds:Inconsistent/DUnknown C r—-

6) sobre vteumb hacemos: `modprobe drbd` para cargar el módulo del kernel, `drbdadm up drbddemo` para inicializar el dispositivo y hacemos `drbdadm – –overwrite-data-of-peer primary drbddemo` para poderlo configurar como primario. Si miramos la configuración con `cat /proc/drbd` veremos entre otros mensajes

1: cs:SyncSource ro:Primary/Secondary ds:UpToDate/Inconsistent C r—— y más abajo [>.....................] sync'ed: 0.4

7) En ambas máquinas veremos el dispositivo */dev/drbd1* pero solo desde aquel que es primario lo podremos montar (no desde el que es secundario). Para ello instalamos las herramientas para crear un *XFS filesystem* con la instrucción `apt-get install xfsprogs` y hacemos sobre el primario (vteumb) `mkfs.xfs /dev/drbd1` y luego podremos hacer montarlo `mount /dev/drb1 /mnt` y podremos copiar un archivo como

`cp /var/log/messages /mnt/test`.

8) Para ejecutar unos pequeños tests podemos hacer sobre vteum: `umount /mnt` y luego `drbdadm secondary drbddemo` y sobre nteum primero debemos instalar XFS `apt-get install xfsprogs` luego `drbdadm primary drbddemo` y finalmente `mount -t xfs /dev/drbd1 /mnt`. Podemos comprobar el contenido con `ls /mnt` y veremos los archivos que copiamos en vteum. Con `cat /proc/drbd` en ambas máquinas veremos el intercambio de roles primario por secundario y viceversa.

9) Un segundo test que podemos hacer es apagar el nodo vteum (secundario) para simular un fallo y veremos con `cat /proc/drbd` que el otro nodo no está disponible (*0: cs:WFConnection ro:Primary/Unknown*). Si copiamos hacemos `cp /mnt/test /mnt/test1` y ponemos en marcha vteum, cuando hace el boot sincronizará los datos y veremos sobre nteum *0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r—* y sobre vteum estarán los archivos sincronizados.

10) Si provocamos un fallo sobre el primario (apagándolo o desconectando la red) y tenemos montado el directorio (es lo que conocemos como *shutdown* no ordenado) no habrá problemas para ir al otro nodo, cambiarlo a primario y montarlo. Si recuperamos el nodo primario que que falló veremos que ambos quedan como secundarios cuando vuelve a estar activo por lo cual lo deberemos poner como primario y lo podremos montar nuevamente y en el secundario veremos que se ha sincronizado y luego en unos segundo volverá a su rol de secundario/primario.

11) Si han cambios durante el tiempo que un nod ha estado desconectado como primario obtendremos un mensaje de "split-brain" para recuperarlo deberemos ejecutar sobre el secundario `drbdadm secondary drbddemo`

y `drbdadm --discard-my-data connect drbddemo` y sobre el primario `drbdadm connect drdbdemo` para conectar los nodos y resolver el conflicto. [27]

1.3.3. DRBD + Heartbeat como NFS de alta disponibilidad

El objetivo de este apartado es mostrar como utilizar DRBD y Heartbeat para generar un clúster NFS que permita tener una copia del NFS y que entre en servicio cuando el servidor primario deje de funcionar. Para ello utilizaremos la misma instalación del apartado anterior pero sobre otro disco en cada nodo (sdc) sobre el cual hemos creado una partición (*/dev/sdc1*) y es importante verificar que tenemos los módulos de DRBD instalados (`lsmod | grep drbd`) [28, 29]. A continuación seguimos los siguientes pasos:

1) Creamos un archivo */etc/drbd.d/demo2.res* con el siguiente contenido:

```
resource myrs {
    protocol C;
    startup { wfc-timeout 0; degr-wfc-timeout 120; }
    disk { on-io-error detach; }
    on nteuma {
        device /dev/drbd2;
        disk /dev/sdc1;
        meta-disk internal;
        address 192.168.1.201:7788;
        }
    on vteumb {
        device /dev/drbd2;
        disk /dev/sdc1;
        meta-disk internal;
        address 192.168.1.202:7788;
        }
    }
```

Donde utilizamos la partición de cada disco sobre cada nodo y generaremos el dispositivo */etc/drbd2*. Es importante que los nombres de las máquinas sea exactamente el que nos da `uname -n` y modificar */etc/hosts*, */etc/resolv.conf* y */etc/hostname* para que las máquinas tengan conectividad entre ellas (y con el exterior) a través del nombre y de la IP. Esto se debe realizar sobre las dos máquinas incluyendo la copia del archivo anterior.

2) También sobre las dos máquinas deberemos ejecutar: `drbdadm create-md myrs` para inicializar el dispositivo, `drbdadm up myrs` para activarlo, `drbdadm syncer myrs` para sincronizarlo. Podremos ver el resultado con la instrucción `cat /proc/drbd` en en cual nos mostrará los dispositivos como "Connected" e inicializados.

3) Sobre el servidor primario ejecutamos

`drbdadm - -overwrite-data-of-peer primary myrs`

para indicarle que sea primario y visualizando el archivo */proc/drbd* veremos el resultado.

4) Finalmente ejecutamos/reiniciamos el servicio con la instrucción `service drbd start|restart` con lo cual tendremos un dispositivo */dev/drbd2* preparado para configurar el sistema de archivo.

5) En este caso utilizaremos LVM ya que permite más flexibilidad para gestionar las particiones pero se podría utilizar */dev/drbd2* como dispositivo de bloques simplemente. Instalamos LVM (`apt-get install lvm2`) y ejecutamos:

```
    pvcreate /dev/drbd2                  Creamos la particón LVM física
    pvdisplay                            Visualizamos
    vgcreate myrs /dev/drbd2             Creamos el grupo llamado myrs
    lvcreate -L 20 M -n web_files myrs   Creo una partición lógica web_files
    lvcreate -L 20 M -n data_files myrs  Creo otra partición lógica data_files
    lvdisplay                            Visualizamos

Con esto deberemos tener disponibles las particiones en /dev/myrs/web_files
y /dev/myrs/data_files.
Con ello creamos el sistema de archivos y lo montamos:
    mkfs.ext4 /dev/myrs/web_files
    mkfs.ext4 /dev/myrs/data_files
    mkdir /data/web-files                          Creamos los punto de montaje
    mkdir /data/data-files
    mount /dev/myrs/web_files /data/web-files      Montamos las particiones
    mount /dev/myrs/data_files /data/data-files
```

6) Ahora deberemos instalar y configurar el servidor NFS (sobre los dos servidores) para que pueda ser gestionado por Heartbeat y exportarlo a los clientes. Para ello ejecutamos (`apt-get install nfs-kernel-server`) y editamos el archivo */etc/exports* con el siguiente contenido:

```
/data/web-files 192.168.1.0/24(rw,async,no_root_squash,no_subtree_check,fsid=1)
/data/data-files 192.168.1.0/24(rw,async,no_root_squash,no_subtree_check,fsid=2)
```

Es importante el valor del parámetro 'fsid' ya que con él los clientes de sistema de archivo sobre el servidor primario sabrán que son los mismo que en el servidor secundario y si el primario queda fuera no se bloquearán esperando que vuelva a estar activo sino que continuarán trabajando con el secundario. Como dejaremos que Heartbeat gestione el NFS lo debemos quitar de *boot* con `update-rc.d -f nfs-common remove` y `update-rc.d -f nfs-kernel-server remove`.

7) Finamente debemos configurar Heartbeat y para ello modificamos el archivo */etc/ha.d/ha.cf* con:

```
autojoin none
auto_failback off
keepalive 2
warntime 5
deadtime 10
initdead 20
bcast eth0
node nteuma
node vteumb
logfile /var/log/ha-log
debugfile /var/log/had-log
```

Se ha indicado 'auto_failback = off' ya que no deseamos que vuelva al original cuando el primario retorne (lo cual podría ser deseable si el hw del primario es mejor que el del secundario en cuyo caso se debería poner a 'on'). 'deadtime' indica que considerá el servidor fuera de servicio luego de 10s, y cada 2s preguntará si están vivos. Dejamos el ficheros */etc/ha.d/authkeys* como ya lo teníamos definido y ejecutamos */usr/share/heartbeat/ha_propagate* para copiar los archivos en el otro servidor (también se podría hacer manualmente).

8) Para indicar una dirección virtual a los clientes NFS usaremos 192.168.1.200 así ellos siempre tendrán esta IP como referente independientemente de quien les esté prestado el servicio. El siguiente paso será realizar la modificación del archivo */etc/ha.d/haresources* para indicarle a Hearbeat cuales son los servicios a gestionar (IPV, DRBD, LVM2, Filesystems y NFS) los cuales los deberemos introducir en el orden que se necesiten:

```
nteum \
IPaddr::192.168.1.200/24/eth0 \
drbddisk::myrs \
lvm2 \
Filesystem::/dev/myrs/web_files::/data/web-files::ext4::nosuid,usrquota,noatime \
Filesystem::/dev/myrs/data_files::/data/data-files::ext4::nosuid,usrquota,noatime \
nfs-common \
nfs-kernel-server
```

Se debe copiar este archivo en los dos servidores (sin modificar) ya que este indica que nteuma es el servidor primario y cuando falle será vteumb tal y como lo hemos explicado en ha.cf.

9) Finalmente podremos iniciar|reiniciar Heartbeat (con la orden `service heartbeat start|restart`) y probar a montar el servidor en una misma red y hacer las pruebas de fallo correspondientes.

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 "Clúster, Cloud y DevOps". 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 y Ganglia extrayendo conclusiones sobre su funcionalidad y prestaciones.

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 (hearbeat, pacemaker, drbd).

9. Con Heartbeat + DRBD crear un sistema de archivos NFS redundante y hacer las pruebas de fallo de red sobre el servidor primario (deshabilitando/habilitando la red) para que el servidor secundario adquiera el control y luego devuelva el control a este cuando el primario recupere la red.

Bibliografía

[1] **Eduardo Ciliendo, Takechika Kunimasa** (2007). *Linux Performance and Tuning Guidelines*.
<http://www.redbooks.ibm.com/redpapers/pdfs/redp4285.pdf>

[2] **Nate Wiger** *Linux Network Tuning for 2013*.
<http://www.nateware.com/linux-network-tuning-for-2013.html>

[3] **Bravo E., D.** (2006). *Mejorando la Performance en Sistemas Linux/Unix*. GbuFDL1.2. <diegobravoestrada@hotmail.com>
<http://es.tldp.org/Tutoriales/doc-tut-performance/perf.pdf>

[4] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.

[5] *Optimización de servidores Linux*.
<http://people.redhat.com/alikins/system_tuning.html>

[6] *Performance Monitoring Tools for Linux*.
<http://www.linuxjournal.com/article.php?sid=2396>

[7] *Monit*.
<http://mmonit.com/monit/>

[8] *Munin*.
<http://munin-monitoring.org/>

[9] *MRTG*.
<http://oss.oetiker.ch/mrtg/>

[10] **M. Rushing** *Fix for MRTG Generating SNMP_Session Error in Debian Wheezy*.
<http://mark.orbum.net/2013/06/07/fix-for-mrtg-generating-snmp_session-error-in-debian-wheezy-and-possibly-ubuntu/>

[11] *SNMP - Debian*.
<https://wiki.debian.org/SNMP>

[12] *Ganglia Monitoring System*.
<http://ganglia.sourceforge.net/>

[13] *Monitorización con SNMP y MRTG*.
<http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance>

[14] **D. Valdez** *Configuración rápida de MRTG*.
<http://sysnotas.blogspot.com.es/2013/06/mrtg-configuracion-rapida-para-debian.html>

[15] Howto *sobre instalación de MRTG en Debian*.
<http://preguntaslinux.org/-howto-instalacion-de-mrtg-monitoreo-debian-t-3061.html>

[16] *Cacti*.
<http://cacti.net/>

[17] *Net-SNMP - MIBs*.
<http://www.net-snmp.org/docs/readmefiles.html>

[18] *Linux-HA*.
<http://linux-ha.org/wiki/Main_Page>

[19] *Documentación de Linux-HA*.
<http://www.linux-ha.org/doc/users-guide/users-guide.html>

[20] *Frysk*.
<http://sources.redhat.com/frysk/>

[21] *Pacemaker documentation*.
<http://clusterlabs.org/doc/>

[22] *Pacemaker Cluster From Scratch*.
<http://clusterlabs.org/doc/en-US/Pacemaker/1.0/pdf/Clusters_from_Scratch/Pacemaker-1.0-Clusters_from_Scratch-en-US.pdf>

[23] **I. Mora Perez**.*Configuring a failover cluster with heartbeat + pacemaker*.
<http://opentodo.net/2012/04/configuring-a-failover-cluster-with-heartbeat-pacemaker/>

[24] **F. Diaz**.*Alta Disponibilidad con Apache2 y Heartbeat en Debian Squeeze*.
<http://www.muspells.net/blog/2011/04/alta-disponibilidad-con-apache2-y-heartbeat-en-debian-squeeze/>

[25] *Pacemarker*.
<http://clusterlabs.org/doc/en-US/Pacemaker/1.0/html/Pacemaker_Explained/s-intro-pacemaker.html>

[26] *DRBD*.
<http://www.drbd.org/home/what-is-drbd/>

[27] *DRBD tests*.
<https://wiki.ubuntu.com/Testing/Cases/UbuntuServer-drbd>

[28] **R. Bergsma**.*Redundant NFS using DRBD+Heartbeat*.
<http://blog.remibergsma.com/2012/09/09/building-a-redundant-pair-of-linux-storage-servers-using-drbd-and-heartbeat/>

[29] **G. Armer**.*Highly Available NFS Cluster on Debian Wheezy*.
<http://sigterm.sh/2014/02/highly-available-nfs-cluster-on-debian-wheezy/>

[30] **Leung, C. T**. *Building a Two-Node Linux Cluster with Heartbeat*.
<http://www.linuxjournal.com/article/5862>

[31] **Majidimehr, A.** (1996). *Optimizing UNIX for Performance*. Prentice Hall.

[32] *Monitor Debian servers with monit*.
<http://www.debian-administration.org/articles/269>

[33] *Monitorización con Munin y monit*.
<http://www.howtoforge.com/server_monitoring_monit_munin>

[34] *Monitorización con SNMP y MRTG*.
<http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance>

[35] *The Enterprise-class Monitoring Solution for Everyone*.
<http://zabbix.com>