

---

## Monitoreando y midiendo el rendimiento de un servidor Postgresql en Sistemas Windows, Linux y Solaris. Parte 1

---

### Consideraciones Preliminares

Antes de comenzar debemos tener en cuenta a que plataformas está destinado este White Paper:

- Sistema Operativo: Linux, Windows. Otros derivados de Unix podrían incluirse.
- Base de Datos: Postgresql 8.1 a 8.3

En este documento, ud. encontrará los caminos para monitorear un servidor Postgresql utilizando las herramientas incorporadas en el sistema operativo más, algunas adicionales que pueden ser descargas gratuitamente desde el sitio [pgdounry.org](http://pgdounry.org) (repositorio oficial de Postgresql).

Para entender de manera cabal este White Paper, le aconsejamos tener conocimientos básicos de programación sobre bash (GNU/Linux y derivados de Unix) y batch en Windows. También deberá conocer muy pequeños conceptos de expresiones regulares en Perl o Python, pero no serán obligatorios en un principio.

### Objetivos de la monitorización

Los fines principales de monitorear un servidor de bases de datos son:

- Verificar el consumo de recursos.
- Actividad positiva del mismo.
- Tiempo de respuesta del servidor.
- Recabar la mayor cantidad de información a fin de poder tener los suficientes datos para ubicar donde esta el problema.
- Detección de problemas de hard o red.
- Obtener información de una determinada tarea o consulta (especial para puesta a punto de sentencias o servidor).

Por lo que deducimos que tenemos tres grupos: actividad del motor, comportamiento del servidor e investigación de consultas.

### Información sobre procesos

Los procesos son tareas que se ejecutan con una cuota de tiempo en el procesador. Cada proceso, puede tener hilos de ejecución internos, simulando tener varios procesos hijos. Los procesos pueden consumir rangos de memoria y tiempos de procesamiento determinados en la configuración del sistema operativo y desde las aplicaciones o herramientas que los lanzan.

Sin embargo, es el SO quien administra de manera transparente al usuario dichas asignaciones (aunque uno pueda asignar determinados retoques desde el SO). Para tales ajustes, es requerido poseer privilegios desde el sistema operativo para poder asignarlas.

Los procesos tienen asignados una *prioridad*. Es esta la que le indica al sistema operativo cuanto tiempo y cantidad de recursos de procesador le asignará. En el caso específico de plataformas Unix o derivadas, este valor puede ir de -20 a 20, siendo la menor, la más prioritaria. En Windows, desde el Administrador de Tareas, `ctfmon.exe` o `tasklist` desde la línea de comandos, podremos observar 6 niveles de prioridades que van desde 'Tiempo Real' a 'Baja'.

Algunos sistemas operativos, tienen mecanismos avanzados de asignación de permisos, tal como Solaris. Este, tiene la opción de FP (Fixed Priority) el cual irá variando la prioridad de acuerdo al uso de los procesos.

Además, podemos considerar el nivel de intrusividad del monitoreo. A mayor cantidad de información recolectada, más se verá afectado el sistema. Sin embargo y por lo general, los monitoreos no suelen ser consumidores abruptos de recursos.

A diferencia de la monitorización normal, existen los *benchmarks*. Estas técnicas, permiten medir el rendimiento de un determinado servidor, ya sea con datos ficticios y aleatorios, como con datos reales de una base. Este tipo de evaluaciones suele consumir muchos recursos, en especial cuando se hacen pruebas de stress sobre los servidores para conocer los límites de prestación del mismo.

–

Existen herramientas específicas en Postgresql para la efectivización de este tipo de tests, entre ellos el más popular es PgBench. Sin embargo, y gracias al auge de los lenguajes de 'scripting', es muy sencillo diseñar un bench que incluya las particularidades de nuestro sistema.

PgBench requiere primero una creación de una relación (pgbench\_branches), por lo que el primer comando a ejecutar es `'pgbench -U<usuario> -i <base>'`. Luego, simplemente quitamos la opción `-i`. Podemos indicar cantidad de conexiones (`-c num`), forma de comprometer una consulta (`-M [extended|simple|prepared]`), cantidad de transacciones por cliente (`-t num`), duración en segundos (`-T num_segundos`), entre otras.

También está disponible un script en Python desarrollado por Mariano Reingart y Emanuel Calvo Franco durante el viaje hacia la ciudad de Junín, cuando se dirigían al PgDay de esa misma ciudad. El código de dicho benchmark está disponible de manera totalmente libre y será publicado en nuestro track (Wiki) prontamente. Sin embargo PgBench es una herramienta muy útil e informativa, por lo que cumple ampliamente sus propósitos.

Para hacer más legible el presente documento, dividiremos el monitoreo de los procesos desde el SO y desde la base, así como también las herramientas específicas para cada plataforma.

Es recomendable, que los monitoreos en sistemas en producción sean constantes, pero no intrusivos (a menos que una situación en especial lo requiera).

Con respecto a los tests, utilizaremos unidades de testeo con herramientas propias de Postgresql.

## Monitoreando en plataformas Linux o Unix compatibles

### Herramientas Previas

Existen una serie de comandos cde los que el usuario debe estar familiarizado con su salida:

- ps

- lsof
- free
- iostat
- mpstat
- sar (si lo tiene)
- htop o top
- vmstat
- /etc/init.d/postgres status o service postgres status (de acuerdo a la distribución)
- netstat
- pgrep
- awk y perl (básico)
- Tuberías y redirecciones.
- Permisos de usuarios (muchas veces si el usuario no tiene permisos, no podrá ver los procesos)

### Comandos

Podemos empezar por verificar que el servidor este corriendo actualmente:

```
postgres@SULLIVAN-2k9:~$ pgrep 'postgres|postmaster' | xargs ps
  PID TTY          STAT TIME   COMMAND
 4526 ?        S      0:00 /usr/local/pgsql/bin/postmaster -D /var/lib/pgsql/data
 4622 ?        Ss     0:02 postgres: writer process
 4623 ?        Ss     0:01 postgres: wal writer process
 4624 ?        Ss     0:00 postgres: autovacuum launcher process
 4625 ?        Ss     0:00 postgres: stats collector process
 4739 ?        Ss     0:00 postgres: ubuntu ubuntu [local] idle
```

Esto indica que el servidor está corriendo. Si observamos en más detalle nos esta diciendo:

- El PPID 4526 nos está informando el proceso principal que se abrió contra el cluster. También nos esta informando la ruta de la ubicación física del PGDATA.
- El PPID 4622 es el proceso de escritura sobre la base.
- El PPID 4623 es el proceso que escribe sobre la WAL.
- El PPID 4624 es el proceso de autovacuum y el 4635 es el recolector de estadísticas. Estos dos procesos puede ser desactivados, pero no es recomendable hasta el momento.
- El PPID 4739 es un cliente abierto. Nos está indicando también sobre que base y que usuario se están utilizando. Incluido a esto, también nos informa que se está accediendo localmente.

## Monitoreando y midiendo el rendimiento de un servidor Postgresql en Sistemas Windows, Linux y Solaris. Parte 1

Viendo más de cerca los procesos, podemos añadir mayores opciones al comando 'ps', para que nos muestre información más detallada.

```
# ps fauxw | egrep 'postmaster|postgres'
```

Esto nos dará información más precisa y extensa, lo que incluye el tiempo de consumo en CPU.

Vamos a ir un paso más allá de los procesos. Una de las cosas más importantes que debemos saber es a que archivos está accediendo un determinado proceso. En el ejemplo siguiente, se decidió obtener el listado de todos los archivos que están siendo accedidos por el proceso de escritura WAL. Se le añadió un filtro 'head' para fines de legibilidad.

```
postgres@SULLIVAN-2k9:~$ pgrep 'postgres|postmaster' | xargs ps \
> | awk '$ ~ "wal" {print $1}' | xargs ls -p | head -2
COMMAND  PID  USER  FD  TYPE DEVICE  SIZE  NODE NAME
postmas 4623 postgres cwd   DIR    3,3   4096 50674586 /var/lib/pgsql/pgsql_8.3
xargs: ls: terminado por la señal 13
```

En el caso de necesitar conocer la cantidad solamente, debemos agregar " | wc -l " .

Saber cuantos archivos consume el servidor de base de datos es importante para tener en cuenta el parámetro del kernel (maxfiles, openfiles). Este parámetro establece el máximo de archivos abiertos. Esto es de suma importancia en ambientes críticos.

Una de las formas con la que podemos saber cuantos archivos está accediendo el servidor postgres, es tecleando:

```
# lsdf | perl -lane 'if (/postgres|postmaster/) {print $F[1]}' | wc -l
```

Esto es simplemente el total de ocurrencias que filtra la expresión regular de Perl. Esta línea no es lo suficientemente inteligente como para diferenciar entre distintos clusters (recordar que podemos tener dos clusters en una máquina, lo que el conteo debería hacerse por separado).

Mejorando la línea anterior podemos obtener la cantidad de procesos activos en un determinado momento a fin de crear estadísticas de conexiones. Simplemente deberemos añadir la siguiente línea de comando en el 'cron' del sistema y este se encargará de loguear las ocurrencias.

```
# ps -A | egrep '(postgres|postmaster)' | wc -l |
xargs echo `date +%Y%m%d%H%M` >> /stat
```

En el cron simplemente podremos establecerlo para correr cada 20 o 10 minutos de acuerdo a nuestras necesidades:

```
*/10 * * * * root <comando>
```

Haciendo el siguiente análisis conoceremos el valor más alto de procesos simultáneos:

```
# awk '{print $2}' </stat | sort -u | head -1
```

Debemos tener en cuenta que hay procesos que son netamente del servidor, por lo que no son conexiones clientes.

Una de las cosas que más afecta el rendimiento de las bases de datos, son los procesos I/O. Por lo tanto, cuanto más logremos disminuir la latencia en estos procesos, menores serán los tiempos de respuesta. Es por eso que siempre es recomendable configurar un RAID para una base de datos.

La herramienta iostat no permitirá ver las estadísticas de escritura y lectura en disco. Para hacerlo más dinámico se recomienda utilizar la herramienta 'watch':

```
# watch --interval=1 'iostat 1 1'
```

La salida de dicho comando se puede apreciar en la siguiente imagen:

```
Linux 2.6.20-15-generic (SULLIVAN-2k9) 13/08/09
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,62    0,00  45,10   0,49    0,00   53,79

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
hda                 1,95         77,09          4,73       325176       19940
hdb                 0,01          0,14          0,00          600          0
hdd                 0,02          0,14          0,00          600          0
```

De este gráfico podemos observar que: el servidor tiene 3 discos (de los cuales 1 es el que está siendo escrito), el porcentaje de iowait (es la espera para poder escribir y leer y a mayor porcentaje, indicará necesidad de contar con más recursos en almacenamiento separados o RAID).

Si bien iostat y lsof son comandos muy útiles, puede que no sean tan intuitivos a primera vista. Existe una herramienta en particular que es muy sencilla y que permite ver los recursos consumidos en I/O por proceso. Es iotop, una herramienta desarrollada en Python y pueden encontrarla en su sitio web <http://guichaz.free.fr/iotop>. Esta herramienta requiere que tengamos Python, kernel superior a 2.6 e instalado el paquete python-pkg-resources (en debian y derivados). Requiere además algunos parámetros en el kernel activados.

Cuando el servidor se queda sin memoria o el proceso de postgres requiere más memoria RAM de la que el SO le puede otorgar a las aplicaciones, el SO comienza a 'swapear' (anglicismo que indica que se comenzó a utilizar el archivo de intercambio para trabajos que no pueden ser ejecutados en RAM por falta de espacio o cuota asignada). Este mecanismo, por lo general, arroja muy malos tiempos de respuesta por lo que debemos evitar en todo momento que el servidor se vea obligado a 'swapear'.

Los comandos free y vmstat nos permitirán ver el espacio libre de memoria RAM y el uso de la partición virtual o Swap.

```

Mem:          total    used    free   shared  buffers   cached
        605000    296120    308880         0         516    201072
-/+ buffers/cache:    94532    510468
Swap:        979956         0    979956

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 308880  516 201072  0  0   34  2 263  73  1 45 54  0
    
```

La imagen anterior fue ejecutada con el comando 'free ; vmstat 1 1'. Se recomienda utilizar watch para que muestre los resultados de manera constante y dinámica.

Del primer párrafo del resultado del comando anterior, nos interesa la línea 'Swap'. Si verificamos el valor 'used' está en 0, por lo que indica que hasta el momento todos los procesos están usando la RAM.

También vemos que el total de la memoria RAM es de 605000 MB, teniendo libres 308880.

EL segundo párrafo corresponde al comando 'vmstat', el cual nos indica que no hay actividad en la swap (si 0, so 0).

Otra de las cosas que nos interesa saber es la actividad del procesador. El comando para esto es 'mpstat'.

```

10:19:02 CPU %user %nice %sys %iowait %irq %soft %steal %idle intr/s
10:19:03 all 0,00 0,00 8,82 0,00 0,98 11,76 0,00 78,43 270,59
Media:  all 0,00 0,00 8,82 0,00 0,98 11,76 0,00 78,43 270,59
    
```

Es recomendable, utilizar 'top' o 'htop' para un resumen de lo más importante de todos estos comandos. Ambas herramientas no están en todos los sistemas unix-like, por lo que es recomendable conocer las herramientas estándar.

Tanto top como htop son muy amigables, por lo que no deberían tener mayores inconvenientes en la lectura de sus resultados.

En el caso de 'top' es aconsejable que cuando se despliegue la ejecución, utilizar la ayuda (presionando 'h') para poder adaptarlo a lo que queremos observar con más detalle. Por ejemplo: Si queremos desplegar la información del tamaño por proceso en el archivo de intercambio y en memoria, debemos presionar 'f' (agrega columnas) y luego 'P' y 'S'.

```

top - 14:05:33 up 47 min, 4 users, load average: 0.62, 0.62, 0.56
Tasks: 96 total, 3 running, 93 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.3%us, 7.9%sy, 0.0%ni, 71.7%id, 0.0%wa, 0.3%hi, 18.8%si, 0.0%st
Mem: 605000k total, 393944k used, 211056k free, 588k buffers
Swap: 979956k total, 0k used, 979956k free, 254188k cached

  PID USER      PR  NI  VIRT  RES  SHR  S %CPU  %MEM    TIME+  SWAP  DATA  COMMAND
 4628 postgres 15   0 71004 1520 896 R  2.0  0.3   0:46.19  67m  872 postgres
 4333 root       15   0 42624 22m 6384 S  1.3  3.8   0:33.75  19m  16m Korg
 4719 emanuel  15   0 36900 5200 2920 S  1.3  0.9   0:23.26  30m 1892 xfce-mcs-manage
 4733 emanuel  15   0 134m 36m 20m S  1.3  6.2   1:23.01  97m  96m firefox-bin
 4743 emanuel  15   0 50108 16m 9284 R  1.3  2.8   0:10.89  32m 7288 xfce4-terminal
   4 root      RT   0  0  0  0 S  0.7  0.0   0:05.28  0  0 watchdog/0
 4626 postgres 15   0 14240 928 320 S  0.7  0.2   0:07.53  13m  892 postgres
 4629 postgres 15   0 71004 1164 540 S  0.7  0.2   0:17.74  68m  872 postgres
 5566 emanuel  15   0 2308 1100 852 R  0.7  0.2   0:00.12 120k 348 top
   5 root      10  -5  0  0  0 S  0.3  0.0   0:03.44  0  0 events/0
 4262 haldaemo 15   0 2160 904 780 S  0.3  0.1   0:11.57 125k 252 hald-addon-stor
    
```

Verificando la apertura del servidor a la red

Hemos configurado el servidor para que pueda escuchar sobre el puerto 5432, pero además le establecimos que pueda escuchar desde otro host (esto último agrega que el puerto no sólo esta creado sino habilitado para recibir y comunicarse con otros clientes en otras terminales).

## Monitoreando y midiendo el rendimiento de un servidor Postgresql en Sistemas Windows, Linux y Solaris. Parte 1

```
emanuel@SULLIVAN-2k9:~$ netstat | grep 5432
unix 3      [ ]      STREAM  CONNECTED  15788   /tmp/.s.PGSQL.5432
unix 3      [ ]      STREAM  CONNECTED  15432   /tmp/.X11-unix/X0
```

En esta imagen apreciamos que el servidor tiene abierto el socket correspondiente.

La forma manual de investigar acerca de los procesos es buscando dentro de la carpeta /proc. En dicha carpeta encontraremos, por PPID, la información de cada proceso.

Por ejemplo:

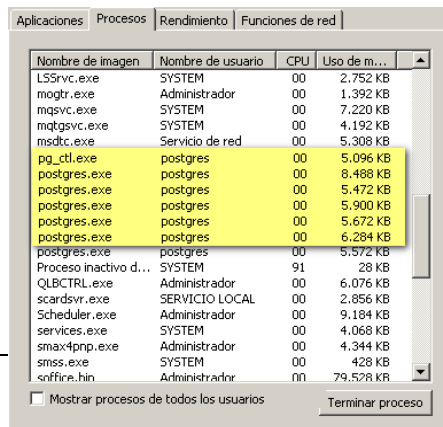
```
cd /proc && pgrep '(postgres|postmaster)' | awk '{print $NF"/stat"}' | xargs cat
```

Esta línea de comando simplemente muestra los contenidos del archivo stat dentro de cada directorio de /proc/<pid>.

Para conocer el contenido de los archivos de /proc, lea el anexo que se adjunta a la documentación.

Otra de las tareas importantes es monitorear el espacio libre que queda en el disco. Si el disco se llena, los datos no se corromperán pero el servidor entrará en el estado PANIC y caerá, por lo que es muy útil tener en cuenta o monitorear constantemente estos valores, en especial si utilizamos PITR o particiones separadas para los archivos de la WAL.

El comando 'df -h' nos mostrará todas las particiones montadas y el espacio que se está utilizando de cada una de ellas. Para saber el tamaño actual del directorio que contiene los archivos de la WAL podemos utilizar 'du -h /ruta/a/la/WAL'.



Nombre de imagen	Nombre de usuario	CPU	Uso de m...
LSrv.exe	SYSTEM	00	2.752 KB
logtr.exe	Administrador	00	1.392 KB
mqsv.exe	SYSTEM	00	7.220 KB
mqtsvc.exe	SYSTEM	00	4.192 KB
msdtc.exe	Servicio de red	00	5.308 KB
pg_ctl.exe	postgres	00	5.096 KB
postgres.exe	postgres	00	8.488 KB
postgres.exe	postgres	00	5.472 KB
postgres.exe	postgres	00	5.900 KB
postgres.exe	postgres	00	5.672 KB
postgres.exe	postgres	00	6.284 KB
postgres.exe	postgres	00	5.572 KB
Proceso inactivo d...	SYSTEM	91	28 KB
QLBCTRL.exe	Administrador	00	6.076 KB
scardsvr.exe	SERVICIO LOCAL	00	2.856 KB
Scheduler.exe	Administrador	00	9.184 KB
services.exe	SYSTEM	00	4.068 KB
smx4pnp.exe	Administrador	00	4.344 KB
smss.exe	SYSTEM	00	428 KB
sniffce.hin	Administrador	00	79.528 KB

### Monitoreando en sistemas Windows

La monitorización en sistemas Windows se puede hacer desde el Administrador de Tareas.

Existen otras herramientas privativas y libres para la medición estadística de los recursos.

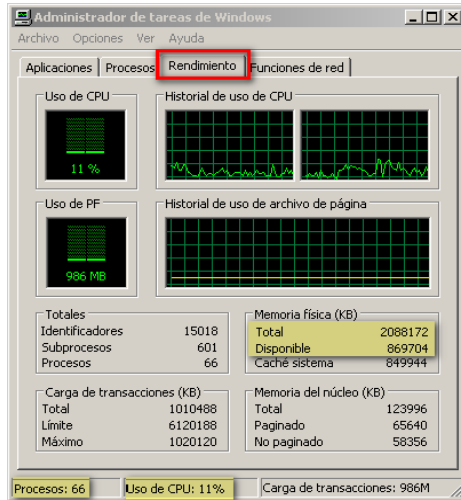
Desde la línea de comando podemos utilizar netstat, tasklist (similar al "ps" de unix-compatibles).

### Utilizando herramientas propias de PgFoundry y nativas del paquete oficial

Como suele ser recomendable, las herramientas específicas de un determinado aplicativo suelen ser más explícitas que las estándares.

Sin utilizar herramientas externas a la empaquetación oficial (dentro de la cual incluiremos PgAdmin III), se puede monitorear la base a través de logs y consultas intensivas al catalogo. De ahí, que es necesario tener establecida la variable de recolección de estadísticas activada (track\_activities).

Como otra recomendación, se encuentra elevar el valor por defecto de la variable del motor default\_statistics\_target. Un valor recomendado para un servidor en producción es superior a 100. Luego de eso, se puede establecer un número mayor o menor de acuerdo a la importancia de la tabla.



este aplicativo, pueden escribirme al mail personal en caso de estar interesados (ya que al momento de escribir este documento, los cambios no fueron aprobados aún por el autor).

La herramienta gráfica más estable para el monitoreo es PgAdmin. Para acceder a esta opción debemos ir al menú "Tools->Server Status". Ahí podremos ver la actividad de las conexiones, los bloqueos y las transacciones. Desde allí mismo podremos seleccionar backends y cancelar sus consultas o terminarlos. Se puede establecer inclusive el intervalo en segundos.

Más allá de las herramientas disponibles para la monitorización "en caliente" de la base de datos, otro método de monitoreo es utilizando el catalogo.

Las tablas-vistas que nos proveen información de escritura y lectura de bloques son:

- pg\_stat\_\*
- pg\_statio\*

Las consultas sobre el catálogo se realizan a través de SQL, y sólo debemos considerar que existen tipos de datos específicos para la relación entre las mismas (OID).

Asimismo poseemos funciones especiales para la administración de los backends, desde al base de datos, permitiendo así, una administración más cabal simplemente desde cualquier cliente.

Seguidamente veremos algunas de las posibles consultas útiles al catalogo

Contabilizar conexiones:  
`SELECT count(*) FROM pg_Stat_activity;`

Verificar hace cuanto se esta ejecutando la última consulta de un backend:

```
SELECT (now() - query_start) as tiempo_query,
       backend_start as back_vivo_desde,
       current_query as query,
       username as usuario
FROM pg_stat_activity;
```

Commits y rollbacks por cada base:

```
SELECT datname, xact_commit, xact_rollback
FROM pg_Stat_database;
```

En cuanto a los logs, por lo general, en un sistema en producción variarán de acuerdo a una situación en especial. El único recaudo a tener es que a medida que el nivel de mensajes que se almacenan son más detallados y a valores más bajos (DEBUG[1-5]) más sobrecarga a nivel de procesamiento y mayor espacio de almacenamiento se verá afectado. Por ello, siempre se recomienda que el almacenamiento de logs se haga separado del almacenamiento de la base y los servidores de aplicación. Esto además permite que el sistema de ficheros a utilizar sea el más sencillo (caso ext2-3, FAT).

La herramienta más popular para el análisis de logs en Postgres es PgFouine. Más detalle en la sección 'Archivado de Logs'.

"lopp" es una herramienta desarrollada en postgres que sirve para medir las estadísticas de i/o por proceso ([http://git.postgresql.org/gitweb?p=iopp.git;a=blob\\_plain;f=iopp.c;hb=HEAD](http://git.postgresql.org/gitweb?p=iopp.git;a=blob_plain;f=iopp.c;hb=HEAD)). Requiere ser compilada con gcc, las indicaciones se hallan en el mismo árbol del git. Asimismo, se requiere tener compilado el kernel de linux con el soporte para el archivo /proc/<pid>/io. Nuevas versiones de kernel, soportan esta opción de manera automática.

Otra herramienta sencilla de instalar es 'pgstat'. Desarrollada en python, puede ser descargada del PgFoundry y sólo requiere descomprimir y ejecutar, indicando a través de parámetros la base y el usuario. Está en versión beta aún, pero es utilizable. Personalmente realicé algunas modificaciones en

## Monitoreando y midiendo el rendimiento de un servidor PostgreSQL en Sistemas Windows, Linux y Solaris. Parte 1

Sumarización de consultas DML:

```
SELECT SUM(n_tup_ins),
       SUM(n_tup_upd), SUM(n_tup_del)
FROM pg_stat_all_tables;
```

Sumarización de los planes:

```
SELECT SUM(seq_scan),
       SUM(seq_tup_read), SUM(idx_scan),
       SUM(idx_tup_fetch)
FROM pg_stat_all_tables;
```

Visualizar bloqueos:

```
SELECT mode, count(mode)
FROM pg_locks
GROUP BY mode ORDER BY mode;
```

Sumarización de I/O en bloques (bloques son de 8k), si desea saber en cantidad de Kb, multiplique por 8:

```
SELECT SUM(heap_blks_read) * 8
FROM pg_statio_user_tables;
SELECT SUM(idx_blks_read)
FROM pg_statio_user_tables;
SELECT SUM(toast_blks_read)
FROM pg_statio_user_tables;
SELECT SUM(tidb_blks_read)
FROM pg_statio_user_tables;
```

Tamaño de las bases:

```
select pg_database_size(name);
select pg_tablespace_size(name);
```

Ver consultas actuales corriendo:

```
SELECT pg_stat_get_backend_pid(s.backendid)
       as procpid,
       pg_stat_get_backend_activity(s.backendid)
       as current_query
FROM
  (SELECT pg_stat_get_backend_idset()
   as backendid) AS s;
```

Uso de disco (cada página es típicamente de 8k):

```
SELECT relfilenode, relpages
FROM pg_class
WHERE relname = 'tabla';
```

Uso del disco para Toast:

```
SELECT relname, relpages
FROM pg_class ,
     (SELECT reltoastrelid
      FROM pg_class
      WHERE relname = 'tabla') ss
WHERE oid = ss.reltoastrelid
      OR oid = (SELECT reltoastid
               FROM pg_class
               WHERE oid = reltoastrelid)
ORDER BY relname;
```

Para indices:

```
SELECT c2.relname, c2.relpages
FROM pg_class c, pg_class c2, pg_index i
WHERE c.relname = 'customer'
      AND c.oid = i.indrelid
      AND c2.oid = i.indexrelid
ORDER BY c2.relname;
```

Encontrar las tablas e índices más grandes (también se puede multiplicar x 8):

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC;
```

Monitorear estado de las tuplas por esquema (preferentemente utilice la opción \x en psql):

```
SELECT schemaname ,
       sum(seq_scan) as Seq_Scan,
       sum(seq_tup_read) as Tuplas_seq_leidas,
       sum(idx_scan) as Indices_scaneados ,
       sum(idx_tup_fetch) as tuplas_x_indices_fetchs,
       sum(n_tup_ins) as tuplas_insertadas,
       sum(n_tup_upd) as tuplas_actualizadas,
       sum(n_tup_del) as tuplas_borradas,
       sum(n_tup_hot_upd) as tuplas_actualizadas_hot,
       sum(n_live_tup) as tuplas_vivas,
       sum(n_dead_tup) as tuplas_muertas
FROM pg_stat_all_tables
WHERE schemaname !~ '^pg.*'
```

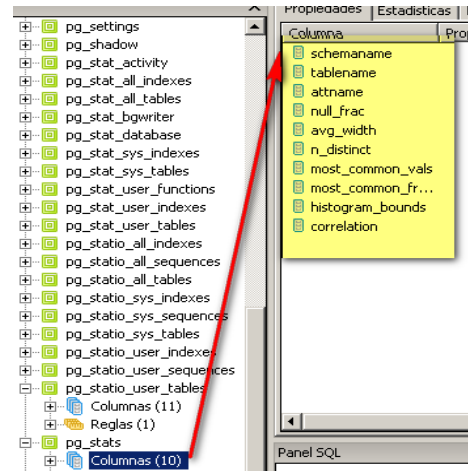
GROUP BY schemaname;

La siguiente consulta obtiene los datos necesarios para calcular los accesos a disco, también podremos obtener una sumariación de las tuplas vivas y muertas de toda la base:

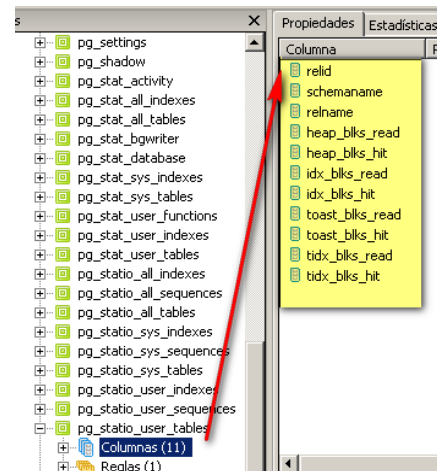
```

SELECT
max(xact_commit) as commits,
max(xact_rollback) as rollbacks,
max(blks_read)::text
|| '/' || (max(blks_read)*8)::text || 'kbs'
as bloques_leidos,
max(blks_hit),
max(numbackends) as numbackends,
sum(seq_scan) as seq_scans,
sum(seq_tup_read) as seq_tup_reads,
sum(idx_scan) as idx_scans,
sum(idx_tup_fetch) as idx_fetchs,
sum(n_tup_ins) as tup_ins,
sum(n_tup_upd) as tup_upds,
sum(n_tup_del) as tup_dels,
max(c.locks) as locks,
max(d.sess) as active,
sum(k.dead) as dead_tup,
sum(k.live) as live_tup
FROM pg_stat_database a, pg_stat_user_tables b,
(SELECT xp.n_dead_tup as dead,
xp.n_live_tup as live
FROM
(SELECT c.relname as nombre
FROM pg_catalog.pg_class c
JOIN pg_catalog.pg_roles r
ON r.oid = c.relowner
LEFT JOIN pg_catalog.pg_namespace n
ON n.oid = c.relnamespace
WHERE c.relkind = 'r'
AND n.nspname <> 'pg_catalog'
AND n.nspname !~ '^pg_toast'
AND pg_catalog.pg_table_is_visible(c.oid)
) xx JOIN pg_stat_all_tables xp
ON xx.nombre = xp.relname
) k,
(SELECT count(*) as locks from pg_locks) c,
(SELECT count(*) as sess
FROM pg_stat_activity
WHERE current_query !~ '!.*<IDLE>*) d
WHERE datname = '<nuestra_base>';
    
```

visualizarla deberemos filtrar a través del campo relname o schemaname. Anteriormente mostramos algunos ejemplos con esta vista.



La vista pg\_stats, contiene las estadísticas por columna, por lo que filtraremos por tabla.



Una consulta como la siguiente:

```

SELECT tablename, attname, avg_width,
    
```

La vista pg\_statio\_user\_tables contiene la información por relación, por lo que si queremos

## Monitoreando y midiendo el rendimiento de un servidor Postgresql en Sistemas Windows, Linux y Solaris. Parte 1

```

n_distinct, most_common_vals,
most_common_freqs,
histogram_bounds, correlation
FROM pg_Stats
WHERE tablename !~ '^(\pg|sql)';

```

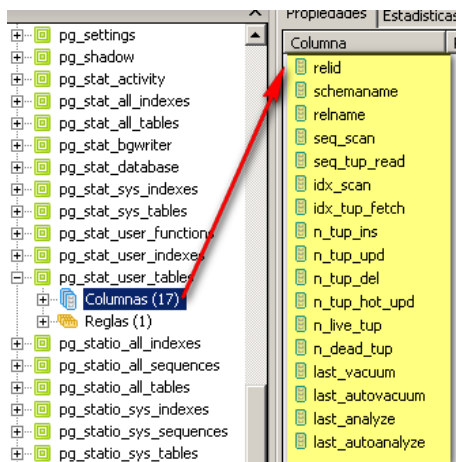
Arroja:

tablename	atname	avg_width	n_distinct	most_commc	most_commc	histogram_bu	correlation
name	name	integer	real	anyarray	real[]	anyarray	real
R	id	4	-1			{1,10,20,30,40}	0.999914
R	valor	35	-1			{0.0021367175,0.000560244}	
tblong	id	4	-0.625	{0}	{0.5}	{Emanuel,Moah}	0.261905
S	id_s	4	-1			{1,10,20,30,40}	1
S	valor_s	43	-1			{*2009-08-18 0:}	-0.0171771
S	tme_s	8	1	{*2009-08-18 0:}	{1}		1
pipi	round	8	2	{1,0}	{0.502,0.498}		0.517145

Esta vista contiene los datos de cada columna, y es importante ya que nos indicará la frecuencia de determinados valores en las columnas, pudiendo inferir directamente en el plan de ejecución. El caso más común es cuando un valor determinado tiene una frecuencia superior a 0.3, en cuyo caso el planeador preferirá accesos por Seq\_scan debido a la repetición de datos.

Esta vista utiliza los valores de la tabla pg\_statistic y es mucho más accesible que esta ultima en términos de consultas.

La vista pg\_stat\_user\_tables posee la siguiente estructura:



Esta vista nos puede mostrar el estado de cada relación, incluidos sus últimos mantenimientos, tuplas

actualizadas, estimación de tuplas muertas y vivas, etc.

También poseemos las vistas para los índices, que nos pueden indicar cuales son los índices más utilizados o, lo que es más útil, los menos utilizados (pg\_stat\_user\_indexes, pg\_statio\_user\_indexes).

Existen una serie de variables que mejoran la recolección de estadísticas (además obviamente de realizar usualmente ANALYZE a la base).

Hay una serie de variables que influyen en la recolección de estadísticas. track\_activities (recolección de estadísticas por sesión), track\_counts (recolección de estadísticas por base) y update\_process\_title (muy útil si realizamos mucho monitoreo desde consola del sistema operativo, actualiza el título del proceso de acuerdo a la consulta) son las principales y por defecto vienen activadas. Posiblemente si tenemos servidores de testeo o desarrollo, queramos desactivarlas para evitar recolección de estadísticas en máquinas que corren varias aplicaciones y que no tienen un rol crucial en el resultado.

Existen otras variables que pueden ser de utilidad también: log\_parser\_stats, log\_planner\_stats, log\_executor\_stats y log\_statement\_stats. Por defecto están en off. La activación de estas variables acarrearán más necesidad de procesamiento del recolector, por lo que afectan directamente al rendimiento del servidor. Si log\_statement\_stats está en on, las otras deben estar en off, debido a que esta contiene las otras. El resto son configuraciones por módulo.

Esta opción genera bastante overhead, así que es recomendable activarla cuando estamos en proceso de afinamiento de determinadas consultas o del servidor.

Otra de ellas es default\_statistics\_target, por defecto está en un valor de 10, pero en producción se recomienda un valor de 100 en producción.

## Archivado de Logs

Los archivos de logs suelen ser molestos porque consumen más I/O a medida que necesitamos más nivel de información.

Sin embargo, tenerlos bajo control es indispensable ya que son los únicos testigos de un error o una situación a la que posiblemente no podamos reproducir fácilmente en un ambiente aislado.

Si vistamos el postgresql.conf observaremos una serie de variables especiales para esta tarea:

`log_destination`: este valor indica a que salida van dirigidos los mensajes.

`logging_collector`: necesario habilitarlo si queremos capturar los mensajes

`log_directory`: directorio en el cual almacenaremos los logs.

`log_filename`: nombre del archivo de log. Este podrá utilizar un formato compatible con POSIX (%Y=año, %m= mes, %d=día , etc.)

`log_truncate_on_rotation`: esto borrará el contenido de un log cuando rota. No es recomendable para ambientes de producción.

`log_rotation_age` y `log_rotation_size`: indican cada cuanto se deberá realizar la rotación de los logs.

`log_min_messages`: esto indica el nivel de logeo.

`log_line_prefix`: esta variable indica como va a estar formateada la salida del log. PgFouine requiere que establezcamos esta variable para analizar el log.

`log_statement`: esto indica que tipo de sentencias irán al log (ddl, mod, all).

Tal como dijimos antes, la herramienta más popular de revisión de logs es PgFouine. Está escrito en PHP, por lo que requiere la instalación del paquete php5-cli (Debian y derivados).

La desventaja que tiene PgFouine es que requiere tocar la salida de los logs, por lo que al principio puede parecer incomodo.

PgFouine tiene varias opciones, las que podremos ver ejecutando 'pgfouine.php' en la linea de comando. La ejecución básica seria : '*pgfouine.php -file <archivo\_log> -logtype stderr*'. La opción logtype deberá corresponder a la que tenemos en el postgresql.conf (`log_destination`). Previo a esto debemos preformatear la salida (`log_line_prefix`) a: '%t [%p]: [%l-1]' si utilizamos stderr.

ANEXO I (link: <http://www.cyberciti.biz/files/linux-kernel/Documentation/filesystems/proc.txt> )