Monitoreo de PostgreSQL con Munin

Gunnar Eyal Wolf Iszaevich

6 de febrero de 2011

Instituto de Investigaciones Económicas, Universidad Nacional Autónoma de México; gwolf@gwolf.org; Cto. Mtro. Mario de la Cueva S/N, Ciudad Universitaria, México D.F. 04510

Resumen

Una de los principales tareas del trabajo diario de un administrador de sistemas, redes o bases de datos es el monitoreo de recursos. Con herramientas adecuadas de monitoreo, el administrador puede no sólo detectar un problema antes de que cause interrupciones en la operación de los sistemas, sino que puede prevenir que ocurran, conociendo la evolución histórica de los valores relevantes.

En el presente trabajo presento el marco de recopilación y graficación de datos *Munin*, enfocándome a su uso para monitorear el gestor de base de datos relacional PostgreSQL, así como en la flexiblilidad y facilidad con que administradores y programadores pueden adecuarlo para desarrollar *plugins* específicos para el monitoreo de sus necesidades puntuales.

Palabras clave: Munin, monitoreo, análisis de tendencias, desarrollo

Abstract

One of the main tasks a system, network or database administrator must face in his everyday work is resource monitoring. Given the adequate tools, an administrator can not only detect a problem before it disrupts systems' operations, but can prevent it from happening, by knowing the historic evolution of the relevant values.

In this work, I am presenting the *Munin* data gathering and graphing framework, focusing on its use for monitoring the PostgreSQL relational database management system, as well a on the flexibility and ease with which administrators and programmers can adequate it to develop specific *plugins* for monitoring their specific needs.

Keywords: Munin, monitoring, tendency analysis, development

1. Introducción

Prácticamente todas las organizaciones –empresas, dependencias de Estado, entidades académicas, etc.– han visto cómo los procesos informáticos van ocupando lugares cada vez más centrales en su operación. Para mantener su operación, se han creado los roles de administradores de sistemas, de redes, de bases de datos, etc.

El trabajo de estos administradores debe ser preventivo, no reactivo — Esto es, un buen administrador de sistemas debe estar atento a los cambios que se presentan en los servicios a su cargo, para poder atacar a los problemas antes de que se presenten. Una de las tareas más importantes que es necesario realizar periódicamente, para encontrar tendencias y anticiparse a los problemas, es el monitoreo de recursos.

El presente trabajo se enfoca en presentar a *Munin* [1], una herramienta orientada a trabajo en red de monitoreo histórico de recursos que puede ayudar –a través de la graficación– a analizar tendencias en su uso, y a bosquejar cómo desarrollar agentes de monitoreo específicos a elementos no contemplados por un desarrollo genérico. Entre sus criterios de diseño se encuentra el ofrecer una instalación muy simple, *plug-and-play*, y al mismo tiempo facilitar a sus usuarios el desarrollo de agentes recolectores de información para adecuarlo a sus necesidades.

Más específicamente: Este trabajo es preparado para su presentación dentro de los trabajos del III PGDay Latinoamericano 2011 [2], razón por la cual haremos énfasis en el uso de Munin para monitorear bases de datos PostgreSQL

Nuestra intención es presentar a Munin detallando algunos casos de uso; para encontrar referencias de uso, administración y desarrollo, sugerimos consultar la documentación oficial del proyecto [3].

2. Arquitectura operativa de Munin

A continuación, delineamos los criterios básicos de desarrollo de Munin. Cabe mencionar que la versión 2.0, actualmente en desarrollo, probablemente revise o retire alguno de los puntos aquí mencionados.

Información histórica La función principal de Munin es presentar un concentrado con la evolución de los diferentes factores que monitoree del sistema, fácil de analizar visualmente, presentando los detalles en vista diaria, semanal, mensual y anual. Una gráfica ejemplo puede apreciarse en la figura 1.

En la figura 2 se aprecia un ejemplo dramático de la importancia de mantener el contexto histórico de la información: Al considerar los niveles de uso del sistema de archivos con una temporalidad local, a través del día o de la semana, la voz de alerta parece ir sobre el uso cercano al 90 % de uno de los sistemas de archivos (indicado por el color verde); sin embargo, al ver la información anual, resultan mucho más alarmantes el crecimiento limeal sostenido del sistema marcado con el color naranja, que aumentó a

Figura 1: Resultado del monitoreo con Munin: Gráfica del día, la semana, el mes y el año de interrupciones y cambios de contexto

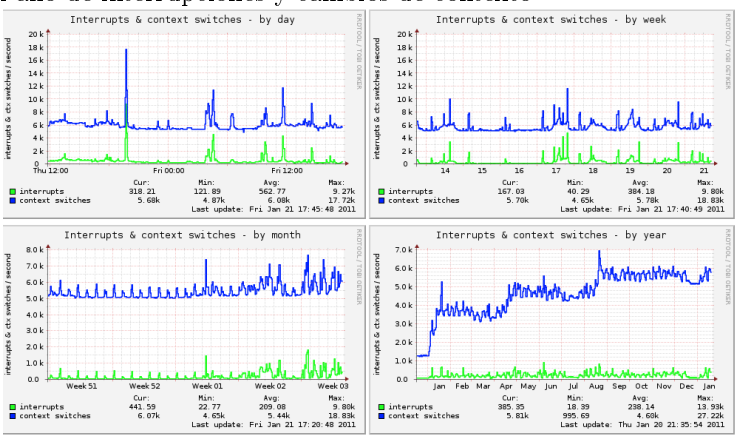
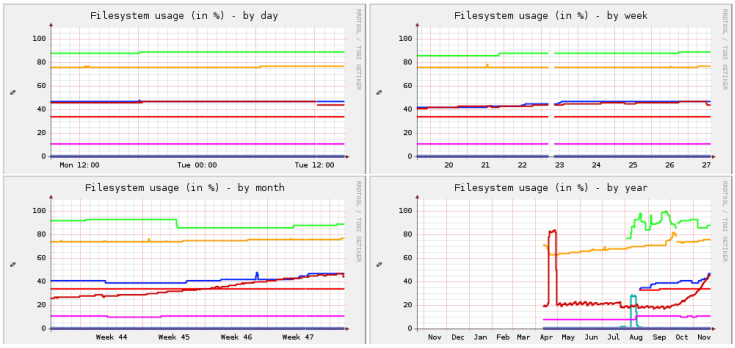


Figura 2: Importancia de la información histórica en las gráficas de niveles del sistema (leyenda recordada para conservar el espacio)



una tasa constante casi un 10 % de utilización a lo largo de seis meses, y más aún, el comportamiento aparentemente exponencial del indicado por el color rojo, que pasó del 20 % en septiembre al 40 % en noviembre.

Archivos estáticos Partiendo del principio de mínima superficie de exposición, y a diferencia de buena parte de los marcos de monitoreo accesibles por Web, consultar los datos recopilados por Munin no requiere de la ejecución de código — Toda la información generada por Munin queda disponible como archivos HTML y gráficas PNG.

Esto significa, claro, que la información presentada está en un formato final — Los niveles de cada una de las gráficas nos es presentada por día, semana, mes y año,

RRDtool Los datos recopilados por Munin son guardados en una base de datos

especializada, *RRDtool* [4] — Una base de datos respaldada en archivos planos, orientada a guardar datos numéricos a intervalos de tiempo fijos y de una duración total predeterminada, lo cual lleva como consecuencia directa que el espacio de almacenamiento de cada base no crece a lo largo del tiempo.
RRDtool nació como parte de MRTG [5], iniciado en 1994, y convertido en un proyecto independiente hacia 2001. Como herencia de éste proyecto, incluye además la funcionalidad de generación de gráficas [6], y ha sido empleado por un gran número de herramientas de monitoreo.

Componentes Munin se divide en tres componentes principales:

Servidor El único proceso que corre constantemente en todas las máquinas monitoreadas. Su función es responder a las solicitudes del cliente, y configurar y llamar a los *plugins*, manejando todos los aspectos relacionados con la comunicación en red. El servidor debe ejecutarse con privilegios de superusuario, para poder lanzar a cada uno de los *plugins* con los privilegios que requieran.
Al servidor veremos que muchas veces se hace referencia también como *munin-node*, al ser el componente que corre en todos los nodos.

Plugins Cada uno de los agentes de recolección de datos que son invocados por munin-node. Dan la información que monitorean, y son también capaces de describir su función y configuración.

Cliente Proceso que corre periódicamente (típicamente cada 5 minutos) desde un nodo central, interrogando a cada uno de los servidores, y generando las páginas Web estáticas.

Autoconfiguración Uno de los atractivos del sistema es que no hace falta invertir tiempo configurándolo para tener una configuración básica adecuada al sistema, con los agentes *plugins* pertinentes configurados. Un *plugin* puede indicar si considera recomendable su autoconfiguración, e incluso con qué parámetros sugiere ser activado.

Es importante, sin embargo, mantener en mente que Munin *no* es un sistema de alertamiento. Al hacer un monitoreo periódico y presentarnos la evolución de los indicadores del sistema a través del tiempo, sería comprensible esperar que Munin funcionara como sistema de alertamiento[1]. Sin embargo, es fundamental recordar que el monitoreo que realiza es *periódico*, consultando los niveles cada cinco minutos. Munin *no hace monitoreo continuo*, sólo en puntos discretos en el tiempo[2].

[1] Y, efectivamente, permite la notificación de ciertos eventos por correo electrónico. Para quien requiera considerar un sistema flexible y completo de monitoreo y alertamiento, sugerimos referirse a Nagios [7].

[2] Hay fuentes de datos que muestran sus resultados como un *acumulado*, como el contador de bytes transmitidos por una interfaz, y otras que muestran sus resultados como una magnitud instantánea, como el nivel de carga del procesador. En caso del primero, Munin sí registrará los picos que se presenten, pero en el segundo esto resulta imposible. Es fundamental comprender la naturaleza de los datos a monitorear para obtener de ellos conclusiones válidas.

2.1. Protocolo de consulta

El protocolo de consulta al servidor de Munin es muy simple, orientado a texto (por lo cual el funcionamiento de un plugin puede monitorearse fácilmente desde la línea de comandos). La interacción básica consiste de los siguientes tres comandos:

list Presenta un listado de los plugins disponibles. Éste listado es básicamente el listado de archivos presentes en el directorio de plugins (típicamente `/etc/munin/plugins`).

config *plugin* Entrega la descripción de los datos a graficar (tanto ante el humano como ante el graficador de RRDtool) y los rangos de datos esperables.

fetch *plugin* Efectúa la consulta a los datos solicitados, regresando los valores obtenidos.

Si bien hay algunos comandos adicionales, para implementar un plugin sencillo basta con implementar este comportamiento.

3. Plugins disponibles para PostgreSQL

Iniciemos el acercamiento al monitoreo específico de PostgreSQL — Primero a través de los plugins que son distribuídos por parte de los desarrolladores oficiales de Munin, seguido por aquellos que forman parte del sitio comunitario Munin Exchange [8]. Después de revisar estas gráficas disponibles, cubriremos cómo desarrollar plugins a la medida.

3.1. Como parte de Munin

Como parte de la versión 1.4.5 de PostgreSQL, publicada en junio del 2010, encontraremos los siguientes plugins, enfocados a monitorear aspectos generales de la *salud*[3] de la instalación de PostgreSQL como un todo, o de bases de datos específicas:

postgres_bgwriter Buffers esperando ser escritos como procesos de fondo

postgres_cache_ Proporción de datos presentes en el cache. Puede manejarse global o por BD.

postgres_checkpoints Número y tipo de checkpoints

postgres_connections_db Número de conexiones por base de datos

postgres_connections_ Estado de las conexiones activas — Activa, en espera, en transacción o desconocida

[3] Aspectos de rendimiento, utilización de disco, correcto funcionamiento de índices, rendimiento promedio/peor caso de las consultas, etc.

Figura 3: Ejemplo de plugin construído con `Munin::Plugin::Pgsql`: `postgres_users`

```
#!/usr/bin/perl
use strict;
use warnings;
use Munin::Plugin::Pgsql;
my $pg = Munin::Plugin::Pgsql->new(
    title  => 'PostgreSQL connections per user',
    info   => 'Number of connections per user',
    vlabel => 'Connections',
    basequery => "SELECT usename,count(*) FROM pg_stat_activity WHERE " .
            "procpid != pg_backend_pid() GROUP BY usename ORDER BY 1",
    configquery => "SELECT DISTINCT usename,usename FROM " .
                   "pg_stat_activity ORDER BY 1");
$pg->Process();
```

postgres_locks_ Número de candados en cada uno de los estados posibles (de renglón/tabla, exclusivo/compartido, etc.)

postgres_querylength_ Tiempo de vida de la consulta o transacción más vieja (más lenta en entregar resultados)

postgres_scans_ La proporción de búsquedas que han sido satisfechas empleando índices o como barridos secuenciales

postgres_size_ Tamaño en disco de una base de datos (entera)

postgres_transactions_ Transacciones por segundo

postgres_tuples_ Acceso a tuplas, separando por tipo de acceso (seqread, idxfetch, inserciones, actualizaciones, actualizaciones en caliente, remociones)

postgres_users Conexiones activas por cada uno de los usuarios del motor

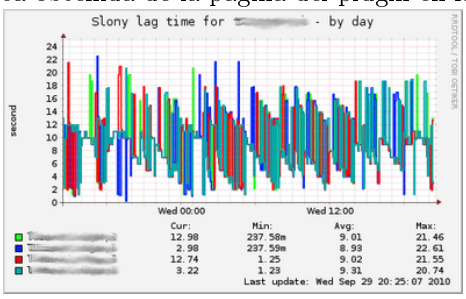
postgres_xlog Tamaño de la bitácora de transacciones

Los plugins cuyo nombre termina en «_» permiten especificar a qué base de datos serán aplicados si son instalados especificando dicho nombre al final — por ejemplo, si instalamos `postgres_connections_template1`, la gráfica mostrará los datos correspondientes a la base `template1`. Si especificamos por nombre «ALL», la mayor parte de estos plugins responderá con los datos agregados entre todas las bases.

3.1.1. Munin::Plugin::Pgsql

Uno de los lenguajes favoritos para la administración de sistemas es Perl [9], mismo que forma parte de la instalación default de prácticamente cualquier sistema Unix moderno. Los plugins descritos en esta sección están implementados empleando el módulo `Munin::Plugin::Pgsql`. Este módulo ofrece una sintaxis declarativa (ver figura 3), con lo que un plugin (construído alrededor de una consulta SQL) puede definirse en menos de diez líneas de código.

Figura 4: Ejemplo de gráfica por plugins de Munin Exchange: Demoras para un cúmulo Slony (gráfica obtenida de la página del plugin en Munin Exchnge)



3.2. Contribuídos por otros usuarios

El sitio comunitario Munin Exchange [8] cuenta con una gran cantidad de plugins con los que se puede complementar los no pocos plugins que forman parte de la distribución oficial — A lo largo de los años, muchos de estos plugins han ido siendo seleccionados para convertirse en parte de la distribución oficial. Varios de los plugins que Munin Exchange nos presentará son ya parte de la distribución oficial.

Al momento de preparación de este texto, son especialmente de llamar la atención los plugins enfocados al monitoreo de cúmulos de nodos del esquema de replicación de PostgreSQL maestro-multiesclavo *Slony* [10] — `slony_view_tag` (ver figura 4), `slony_lag_time` (global) y `slony_lag_time_` (por BD).

Claro está, en Munin Exchange están disponibles también plugins para todo tipo, enfocados a virtualmente cualquier área de la administración de sistemas.

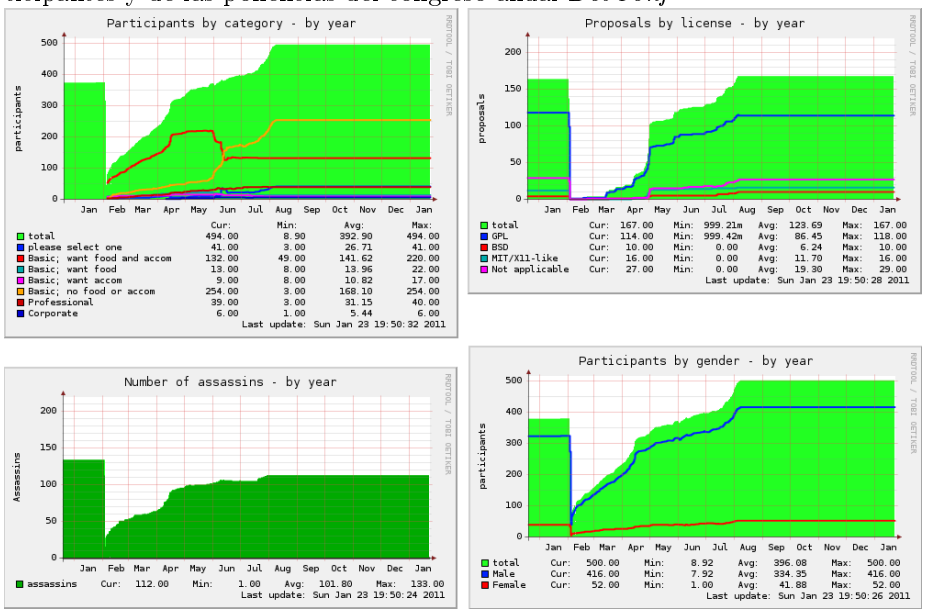
Instalar los plugins puestos a disposición a través de Munin Exchange sólamente implica colocar su código en un lugar adecuado del sistema de archivos[4] (puede ser bajo `/usr/share/munin/plugins`, con el resto de los plugins del sistema, en la jerarquía `/usr/local`, o donde el administrador lo adecuado a sus políticas, crear una *liga simbólica* desde el directorio de configración de Munin (`/etc/munin/plugins`), posiblemente agregar un bloque de configuración (en `/etc/munin/plugin-conf.d`), y reiniciar el proceso servidor (`munin-node`).

4. Desarrollo de *plugins* específicos

El verdadero poder de análisis que otorga Munin va más allá, sin embargo, del monitoreo de los *signos vitales* del servidor. Como observamos en la figura 3, es muy sencillo emplear una consulta SQL para crear un plugin que grafique los resultados de la razón de ser de nuestras bases de datos: Los datos mismos. Para algunos ejemplos de ésto, refiérase a la figura 5.

[4]Las ubicaciones aquí descritas corresponden a la instalación efectuada en un sistema Debian, así como a sus distribuciones derivadas. En otros sistemas, la ubicación de dichos directorios puede cambiar.

Figura 5: Ejemplo de monitoreo a datos del usuario: Diversos datos de los participantes y de las ponencias del congreso anual *DebConf*



Ahora bien, aunque el desarrollo de plugins se simplifica fuertemente al emplear el lenguaje Perl por la disponibilidad del módulo `Munin::Plugin::Pgsql` (ver secc. 3.1.1), éstos pueden ser generados en el lenguaje que para cada tarea prefiera el administrador — El protocolo de interrogación de Munin es suficientemente simple como para que implementarlo no represente limitante alguna.

5. Conclusiones

Un esquema de monitoreo es una herramienta fundamental que todo administrador debe saber emplear. Munin presenta uno de los enfoques más flexibles a esta necesidad, en buena medida gracias a su arquitectura de plugins y a la sencillez del protocolo que emplea para comunicarse con ellos.

A los muchos plugins ofrecidos por los desarrolladores núcleo de Munin se pueden agregar una enorme cantidad adicional, desarrollados por sus usuarios y puestos a disposición en el sitio de intercambio comunitario Munin Exchange. Sin embargo, a juicio del autor, el mayor potencial de aprovechamiento de Munin se alcanza al desarrollar cada administrador los plugins específicos a la naturaleza de su instalación — y más aún, a sus datos mismos.

Referencias

[1] Proyecto Munin; http://munin-monitoring.org/

[2] III PGDay Latinoamericano 2011; http://postgresql.uci.cu/node/22; Universidad de las Ciencias Informáticas (UCI), Comunidad Técnica Cubana de PostgreSQL, Empresa de Telecomunicaciones de Cuba, Centro para la Formación y el Desarrollo del Capital Humano, Comunidad Internacional PostgreSQL

[3] Munin Documentation; http://munin-monitoring.org/wiki/Documentation

[4] *Round Robin Databese tool* — Herramienta de base de datos circular; http://oss.oetiker.ch/rrdtool/

[5] Multi Router Traffic Grapher, http://oss.oetiker.ch/mrtg/

[6] *Creación de gráficas con RRDtool*, Jesús Roncero Franco; Bulma.net, 2002 http://bulma.net/body.phtml?nIdNoticia=1284

[7] Infraestructura de monitoreo y alertamiento *Nagios*; http://www.nagios.org/

[8] *Munin Exchange*, Depósito de plugins de Munin desarrollados por sus usuarios — http://exchange.munin-monitoring.org/

[9] Lenguaje de programación *Perl*; http://www.perl.org/

[10] Infraestructura de replicación *Slony*; http://www.slony.info/