Plan de acción frente a demoras en Oruga Amarilla

Carlos Thompson

Oruga Amarilla, una marca Invermeq e Interlecto

Diciembre de 2013

Resumen

Plan de trabajo para lograr mejoras substanciales en la página *http://test.orugaamarilla.com* de Oruga Amarilla, la cual, en la actualidad, se cuelga frente a algunas consultas.

Este plan de trabajo incluye optimizar la base de datos, crear sistemas de control de procesos en la base de datos y sugerir algunos pasos para una solución más permanente.

Contenido

Capítulo 1

El problema actual

Sección 1.1. Observación

Cuando se intentan cargar páginas que muestran los resultados de algunos sitios, principalmente el sitio San Pedro de Elite Flowers la página tarda un tiempo largo sin presentar ningún tipo de actividad tras el cual aparece un error 500 Internal Server Error.

Consultas subsiguientes, incluyendo aquellas que no pidan información de San Pedro, fallan igualmente.

Sección 1.2. Diagnóstico

Las consultas por últimas actividades de algunos de los sitios, y en particular de San Pedro, ejecutan lentamente en el servidor de bases de datos MySQL *mysql.orugaamarilla.com* .

Estas consultas tardan significativamente más que el tiempo configurado por el servidor web para presentar resultados. Como el programa no termina de ejecutar en el tiempo estipulado, el servidor emite un error 500 Internal Server Error y detiene la ejecución del *script* pero la consulta en el servidor de base de datos sigue ejecutando.

Las subsiguientes consultas, incluyendo aquellas que son generadas por el usuario cuando ejecuta F5 o cambia manualmente el URL de la aplicación frente a la frustración por no recibir información. Este comportamiento justificado del usuario agrava la situación pues encola nuevas consultas.

Este bloqueo en el servidor de bases de datos podría estar causando también pérdidas en la información que la base de datos recibe automáticamente de los sitios. Esto último no ha sido verificado, pero es una conclusión razonable.

Sección 1.3. Posibles soluciones

Entre las soluciones planteadas se incluyen:

1. Optimizar la base de datos con el objeto de reducir el tamaño de las tablas y, en consecuencia, agilizar las consultas individuales.

   (ya se han adelantado pasos en este sentido pero no se han realizado pruebas.)

2. Agregar sistemas de limpieza de consultas abandonadas. Esto es llevar un control de las consultas que se inician pero que no terminan porque el *script* fue suspendido bien por *timeout* del servidor, bien por acción del usuario (detener carga de la página, relanzar la consulta, cambiar la consulta, etc.)

   (Se ha probado con la inclusión de conexiones persistentes, el cual incluye automáticamente un módulo de limpieza, pero no se han observado resultados positivos, igualmente esto sólo serviría para consultas relanzadas o cambio de consulta y no para otro tipo de abandonos.)

3. Agregar rutinas en el diseño de la aplicación web que permita la carga de la página antes de terminar la consulta y que el resultado de la consulta se agregue después, independientemente del tiempo que tarde la consulta.

   Esto tiene un efecto más psicológico pues el usuario no *ve* una página bloqueada sino que es informado que la consulta está tardando y evita la aparición de errores 500 Internal Server Error, disminuyendo por ende la sobrecarga del servidor por parte de usuarios frustrados.

   (La técnica es conocida a través de AJAX, pero aun no se ha buscado cómo incluirla en la página de Oruga Amarilla.)

4. Utilizar un servidor dedicado.

   Varios de los problemas de desempeño son causados por el hecho de que los servidores actuales de Oruga Amarilla son un servidor web compartido y un servidor MySQL compartido.

   Además del desempeño, no se tiene acceso de súper-usuario al servidor lo que impide hacer algunas maniobras de corrección rápidas.

Capítulo 2

Optimización de la base de datos

Aquí se tratarán las estrategias de optimización de la base de datos de Oruga Amarilla.

Sección 2.1. Situación actual

Actualmente la aplicación de Oruga Amarilla utiliza dos tablas en la base de datos `orugadata` en el servidor *mysql.orugaamarilla.com*. Estas tablas son: `dl_status_i` y `dl_status_p`. Actualmente estas tablas poseen respectivamente más de 3,4 y más de 22 millones de registros, ocupando respectivamente más de 400 MB y 1,5 GB.

Toda consulta sobre estados debe hacer un cruce entre estas dos tablas. Desplegar información además requiere consultar otras tablas en las que se almacena la información *no cambiante*.

El tiempo de proceso que se requiere para hacer el cruce de estas tablas, debido al tamaño, es grande. Ya hay ciertas optimizaciones al filtrar los resultados pero esto requiere un ordenamiento el cual también tarda.

Sólo San Pedro compone () registros en `dl_status_i` y un número proporcional de datos en `dl_status_p`.

Sección 2.2. Sugerencia de optimización

En lugar de dos tablas que incluyan todas las estaciones en una consulta cruzada se sugiere crear cuatro o cinco tablas por cada estación, evitando también las consultas cruzadas.

Cada estación tendrá una tabla de datos activos, en los cuales se registren en una sola tabla todas las actualizaciones de los últimos 60 días. (Podrían ser dos tablas, una para estado de instrumentos y otra para alarmas.)

Adicionalmente habría tres tablas de resultados históricos: una tabla con todos los datos anteriores a 60 días y dos tablas con resultados promediados. De estas tablas se eliminarían los datos que se sabe que no son relevantes.

2.2.1. Ventajas.

- Las tablas activas (últimos 60 días y promedios) serán más pequeñas y requerirán menos recursos y menos tiempo en ser consultadas.
- Ninguna de estas tablas requerirá cruce de tablas para hacer las consultas pertinentes lo que reduce el procesamiento en el servidor con la consiguiente ganancia en tiempo.
- La mayor parte de los datos históricos que se requieren se requieren promediados. Las tablas de resultados promediados agilizan estas consultas en la mayor parte de los casos.
- Se conservan los datos históricos relevantes.
- No se guarda información poco relevante.

2.2.2. Desventajas.

- Aumenta la complejidad de las relaciones en la base de datos.
- Cada nuevo sitio implica la creación de nuevas tablas.
- Se duplica información.
- Cierta información se perdería.
- La migración periódica de datos de la tabla activa a la histórica y mantener actualizadas las tablas promediadas requieren de procesos.
- Migrar requerirá de consultas largas.
- Requiere cambios grandes en la aplicación web para responder al nuevo diseño.
- El rediseño planteado podría no ser el óptimo

Ninguna de estas desventajas es crítica. Las desventajas más críticas afectan sobre todo la fase de pruebas y migración las cuales, frente a la situación actual, no representan una desventaja adicional.

Con respecto al tiempo de proceso requerido para mantener las tablas históricas y promediada se sugiere que estas actualizaciones se hagan en consultas diarias. Un lapso de 24 horas no es crítico para borrar datos de la tabla activa y pasarlo a la histórica y es un buen compromiso para mantener al día las tablas promediadas.

2.2.3. Conclusión. Este u otro rediseño de la base de datos **debe** hacerse.

Aun si el rediseño aquí propuesto no es el óptimo debe ser una mejora significativa frente a la situación actual.

Sección 2.3. Avance en el plan

A la fecha (9 de diciembre) se han realizado esquemas de creación automatizada del nuevo diseño de tablas y migración de los datos, pero no se han elaborado los *script*s de la aplicación web que utilicen estos diseños, ni se han hecho pruebas para saber qué tanto mejoran los tiempos de consulta.

Capítulo 3

Optimización en el diseño de la aplicación Web

Aquí se discuten las estrategias de optimización en el diseño de la aplicación web.

Sección 3.1. Situación actual

Actualmente la aplicación web funciona de una forma directa:

1. El servidor recibe la consulta.
2. El *script* en el servidor interpreta la consulta y determina que acciones debe ejecutar antes de desplegar la página.
3. Si se requiere hacer consultas a la base de datos se envían las consultas y se esperan los resultados.
4. (se espera el resultado de cada consulta antes de continuar con el código.)
5. Cuando se ha terminado de consultar, se forma la página.
6. Se devuelve la página al cliente que envió la consulta web.
7. Se termina el proceso.

En esta situación el cliente (el usuario) no recibe información alguna antes de que el *script* en el servidor termine de ejecutar completamente. Si el *script* tarda 20 s en ejecutar, el usuario percibirá una demora de 20 s en la carga de la página. De acuerdo a los estudios de usabilidad, más de 5 s sin ningún tipo de actualización se considera inaceptable.

Si la demora en la ejecución del *script* pasa el tiempo estipulado de inactividad del servidor (*timeout*) [por defecto 30 s, actualmente ampliado a 90 s] el servidor suspenderá la ejecución del *script* y enviará un error 500 Internal Server Error. Este mensaje no es muy útil para el usuario (además que demora en aparecer).

Adicionalmente, cuando se suspende la ejecución del *script* por acción del usuario (cerrar página, relanzar [F5], cambiar de URL) o del servidor web (*timeout*), la consulta en la base de datos sigue activa, y el tiempo que tarda en completar se suma a la siguiente consulta.

Aun si se optimizan las consultas en la base de datos, siempre es posible que alguna consulta tarde más de 5 s en ejecutar, por lo que se recomienda atacar este problema independientemente de las demás soluciones.

Sección 3.2. Sugerencia de optimización

La aplicación web debe rediseñarse para que despliegue información relevante antes de que las consultas a la base de datos culminen.

El flujo sería el siguiente:

1. El servidor recibe la consulta.
2. El *script* en el servidor interpreta la consulta y determina que acciones debe ejecutar antes de desplegar la página.
3. Si se requiere hacer consultas a la base de datos se envían las consultas en procesos independientes.
4. (No se esperan los resultados, es *script* de despliegue de página sigue ejecutando sin esperar el resultado).
5. Se forma la página con los contenidos existentes.
6. Se devuelve la página al cliente que envió la consulta web.
7. Se termina este proceso.
8. Cuando culmina cada proceso paralelo, éste envía una notificación a la página y reemplaza el contenido relevante.

Para este último paso se requiere la elaboración de scrips en AJAX, una extensión de JavaScript.

3.2.1. Ventajas.

1. Se despliega el contenido estático de la página de una forma más rápida, dando a entender al usuario que el servidor está activo.
2. Las consultas pueden tardar el tiempo que sea necesario para ejecutar sin generar *timeouts* en el servidor web.

3.2.2. Desventajas.

1. Es necesario cargar las extensiones de AJAX.
2. El tiempo de ejecución final aumenta, tanto por la carga de AJAX y el tiempo de procesamiento tanto en el servidor como en el cliente.

3.2.3. Conclusión. Al igual que el punto anterior las ventajas superan las desventajas y las desventajas no son relevantes frente a la situación actual.

Sección 3.3. Avance en el plan

Este punto aún no ha sido probado. No existen avances.

Capítulo 4

Sistemas de control de procesos

Uno de los inconvenientes que presenta actualmente la página de Oruga Amarilla son los procesos abandonados en el servidor MySQL.

El servidor MySQL es independiente al servidor web y por lo tanto no es consiente del estado del servidor web.

Sección 4.1. Situación actual

Cuando la consulta a la aplicación web requiere una consulta a la base de datos, el *script* en el servidor web envía la consulta y espera la respuesta.

Si el *script* es suspendido por acción del servidor web (por ejemplo por que se excede el tiempo estipulado de conexión [*timeout*]) o por acción del usuario (cierre del navegador o la pestaña, detención de la carga de la página, relanzamiento [F5], cambio del URL en la barra de navegación, etc.) la consulta en el servidor MySQL sigue activa.

No hay forma en que el *script* en el servidor web mate los procesos pendientes en el servidor MySQL porque el *script* en el servidor web está suspendido.

Como resultado, una consulta que tarda demasiado en ejecutar, seguirá afectando las consultas consiguientes aún cuando las nuevas consultas no tardaran tanto.

Sección 4.2. Sugerencia de optimización

Hay dos acciones que deben tomarse aquí.

La primera es que una nueva ejecución del *script* mate los procesos pendientes de la ejecución anterior.

La segunda consiste en detectar la suspensión del *script* para ordenar labores de limpieza en los procesos que se ejecutan en el servidor de bases de datos.

Sección 4.3. Limpieza de procesos con nueva ejecución de la página

Esta optimización afecta exclusivamente cuando el proceso actual del *script* es suspendido por una acción del usuario tal como relanzar la página o solicitar una nueva página al servidor.

Muchos de los casos serían cubiertos por el siguiente arreglo sugerido, pero la diferencia en cómo se enfrentan los hace complementarios.

4.3.1. Ventajas.

1. La solución es exclusivamente en PHP, no requiere de JavaScript ni AJAX.
2. Es independiente a cualquier otra optmización. No requiere replantear ni el diseño de la aplicación web ni el diseño de la base de datos.
3. Las nuevas consultas no dependen de si la consulta anterior concluyó con éxito. Las nuevas consultas no tienen que esperar a que la consulta anterior haya concluido.

4.3.2. Desventajas.

1. No cubre la totalidad de los casos en los cuales se puede presentar procesos abandonados.
2. Aumenta las labores que debe ejecutar el *script* aumentando el tiempo de ejecución.
3. Debe tenerse cuidado si se abre una nueva pestaña o ventana bajo la misma sesión para que esta no mate los procesos de la ventana anterior todavía activa.

Sección 4.4. Limpieza de procesos tras abandono de página

4.4.1. Ventajas.

1. Cubre la mayoría de los casos en los cuales puede presentarse *scripts* abandonados.

2. Se liberan los recursos una vez se haya abandonado o terminado la ejecución del script, independientemente de si se llama un nuevo script o no.

4.4.2. Desventajas.

1. Tal cual está diseñada la aplicación web no sería útil: la detección de abandono de página sólo cargaría cuando se ha enviado la página y sus archivos asociados al cliente.

   Esta solución debe ir en conjunto con la solución del capítulo anterior: Optimización del diseño de la aplicación web.

2. En conjunto con la Optimización del diseño de la aplicación web debe ejecutarse con cuidado para que no se envíe un mensaje de terminación antes de que carguen los contenidos en caso de terminación normal del *script* sin abandono de página.

4.4.3. Conclusión. Ambas optimizaciones deben llevarse a cabo pues son complementarias.

Ambas optimizaciones requieren de procesos de depuración para que no exista conflicto entre ellas, ni conflicto entre diferentes instancias de una misma sesión, ni conflicto con las otras optimizaciones de diseño web.

Sección 4.5. Avance en el plan

Ya se implementó la modalidad de conexiones persistentes que, en teoría, soluciona automáticamente parte de la limpieza de procesos en el servidor web tras una nueva ejecución de la página, pero las pruebas no indican una mejora significativa.

Capítulo 5

Migración del sistema

Sección 5.1. Situación actual

La pàgina de Oruga Amarilla está actualmente trabajando en un servidor compartido de DreamHost, fisicamente localizado en California, EE. UU. y en el cual los administradores de Oruga Amarilla no tienen acceso de súper-usuario.

Esto aplica tanto para el servidor web como para el servidor de bases de datos.

Al ser un servidor compartido los procesos de Oruga Amarilla están copitiendo en recursos con otros servicios de otras aplicaciones web.

No existen SLA (acuerdos de nivel de servicio) con DreamHost sobre tiempo en línea ni sobre tiempo y velocidad de procesamiento.

Sección 5.2. Sugerencia de optimización

Se sugiere tener un servidor dedicado el cual puede venir en cualquiera de las siguientes modalidades:

1. Alguiler de un servidor virtual dedicado en un servicio de *hosting*.

   Este puede venir a su vez en modalidad gestionada o no gestionada.

2. Alquiler de un servidor dedicado físico en un servicio de *hosting*.

   Este puede venir a su vez en modalidad gestionada o no gestionada.

3. Comprar un servidor.

5.2.1. Ventajas. Adquirir un servidor dedicado, independientemente de la modalidad, trae las siguientes ventajas:

1.

Más adelante se mencionan ventajas y desventajas de cada modalidad.

5.2.2. Desventajas. Adquirir un servidor dedicado, independientemente de la modalidad, trae las siguientes desventajas:

1.

Más adelante se mencionan ventajas y desventajas de cada modalidad.

Sección 5.3. Servidor alquilado gestionado o no gestionado

(Por servidor alquilado nos referimos a un servidor físicamente hospedado en las instalaciones de un servicio de *hosting*, indpendientemente de la modalidad de propiedad del mismo.)

Un servidor gestionado es una modalidad de alquiler mediante la cual la empresa que realiza el servicio de *hosting* se encarga del mantenimiento del servidor y la empresa contratante (v. g. Invermeq) no tiene acceso de súper-usuario (administrador).

Un servidor no gestionado, la empresa de *hosting* sólo se encarga de mantener el servidor conectado tanto a potencia eléctrica como a la red, así como de realizar reinicios físicos cuando el servidor esté bloqueado, pero la administración recae directamente en la empresa contratante.

5.3.1. Ventajas del servicio gestionado.

1. Menor carga administrativa para Oruga Amarilla.
2. Mayor tiempo de servicio. Errores de administración por Oruga Amarilla pueden ser corregidos por los administradores del servicio de *hosting*.
3. Acceso de súper-usuario (*root* o administrador de sistema) restringido que evita hacer cambios y actualizaciones de bajo nivel que puedan afectar negativamente el servicio por errores de procedimiento.

5.3.2. Ventajas del servicio no gestionado.

1. Precios más bajos de alquiler, pues no se está contratando un servicio de administración.

2. Acceso de súper-usuario (*root* o administrador de sistema) que permite hacer cambios y actualizaciones de bajo nivel sin estar elevando tiquetes de servicio.

Sección 5.4. Servidor alquilado virtual o exclusivo

(Por servidor alquilado nos referimos a un servidor físicamente hospedado en las instalaciones de un servicio de *hosting*, indpendientemente de la modalidad de propiedad del mismo.)

Un servidor virtual es un proceso que corre dentro de una máquina que puede prestar otros servicios de virtualización. Se diferencia de un servidor compartido en que la máquina virtual tiene tiempo de procesamiento y espacio reservado.

5.4.1. Ventajas de los servidores virtuales.

1. Aun en una modalidad no gestionada, la empresa de *hosting* contratada realiza labores de mantenimiento periódico con mayor frecuencia que en un servidor no virtual.

2. Es más fácil escalar cuando sea necesario o reducir los recursos requeridos si no se necesitan.

3. En consecuencia, los costos pueden ajustarse a las necesidades.

4. Se pueden separar el servidor web y el servidor de bases de datos en máquinas virtuales diferentes. En el caso de servidores físicos se requerirían dos servidores físicos o un sólo servidor asumiría ambas tareas.

5. En muchas ocasiones la máquina virtual puede hacer uso de más recursos del servidor que aquellos reservados si las otras máquinas virtuales no están haciendo uso de los mismos.

6. Suelen ser más económicos.

5.4.2. Ventajas de los servidores exclusivos.

1. El tiempo de procesamiento es exclusivo. La virtualización no consume recursos.

2. Varios servidores dedicados en un mismo servidor físico tienen igual que compartir recursos de red de último metro. Un servidor físico tiene estos recursos en exclusiva.

3. Aunque la virtualización suele evitar el problema de un servicio compartido de que un proceso de un usuario interfiera con los demás, un servidor de máquinas virtuales no está completamente exento de que un proceso mal ejecutado en una máquina virtual afecte a todo el servidor.

   Esto no sucedería en un servidor físico.

4. Un servidor exclusivo, bien dimensionado, puede salir más económico que un servidor virtual de las mismas características.

Sección 5.5. Servidor alquilado o servidor propio

Por servidor alquilado entendemos un servidor que está físicamente hospedado en las instalaciones de un servicio de *hosting*, independientemente de la modalidad de propiedad del hardware.

Análogamente un servidor propio nos referimos a un servidor físicamente localizado en espacios controlados por Invermeq o alguno de sus asociados, bien sea que haya sido comprado o adquirido en modalidad de liesing.

5.5.1. Ventajas de un servidor alquilado.

1. El servicio de *hosting* contratado se encarga de mantener la conexión del servidor tanto a potencia eléctrica como a la red bajo los términos de un acuerdo de nivel de servicios (SLA).

   Con un servidor propio es necesario gestionar en casa tanto la conexión a Internet como los sistemas de redundancia de energía.

2. El CAPEX es más bajo.

3. Hay mayor flexibilidad si se requiere cambiar de servidor o cambiar las características del servidor.

5.5.2. Ventajas de un servidor propio.

1. El OPEX podría ser mas bajo. (Aunque esto es relativo, ya que los OPEX en EE. UU. son más bajos que en Colombia y aun con un servicio alquilado en Colombia, un operador de *hosting* puede negociar tarifas en bloque.)

2. Propiedad completa, exclusiva y segura de los datos. No hay terceros involucrados que puedan cambiar unilateralmente sus políticas de privacidad ni dar cumplimiento sin aviso de órdenes judiciales.

Sección 5.6. Conclusión

Creo que es conveniente buscar una modalidad de servidor dedicado en un tiempo pronto.

Mientras podamos medir nuestras necesidades reales un servidor virtual alquilado parece ser la mejor opción. En principio creo que un servicio no gestionado es manejable.

Aun así, sin lograr las otras optimizaciones sugeridas, aumentar el poder de cómputo no es una solución real. Este proceso debería esperar a que se terminen las otras optimizaciones.

Sección 5.7. Avance en el plan

Aun no se ha averiguado sobre las modalidades de servidor dedicado aquí planteadas.

Capítulo 6

Conclusiones

Todos las mejoras aquí propuestas incluirán tiempos de baja del servicio así como desarrollo y pruebas.

Durante los tiempos de baja se afectarán tanto los servicios al usuario como los servicios automáticos de recolección de datos, sin embargo estos no afectan la recolección de datos local en cada sitio remoto.

Debe contarse con estos tiempos de baja del sistema que se espera redunden en un servicio más estable.