**3 de Junio de 2016**

**Análisis de rendimiento de una aplicación web (LyricsBox) con JMeter**

**Materia: Programación Distribuida I**

**Alumnos: Eric Loza y Matias Martinho**

**Universidad Nacional de Avellaneda**

**Rendimiento de la API:**

Se realizará un análisis sobre el rendimiento del programa hecho en la primera parte del trabajo práctico. Éste está hecho en PHP usando el framework CodeIgniter y su motor de base de datos es SQLITE. La herramienta utilizada para realizar los test de carga es JMeter. Analizando estos resultados es posible encontrar los diferentes problemas, por ejemplo cuellos de botella, que afectan el normal funcionamiento del programa en un entorno donde muchos usuarios lo utilizan.

**LoadTest en JMeter:**

*En primer lugar, se realizará un test de carga sobre la base de datos directamente.*

Se cargaron datos aleatorios sobre las tablas de la base de datos para poder simular un escenario posible.

Las tablas poseen la siguiente cantidad de registros:

* Cancion: 10000
* Fragmento: 40000
* Juego: 40000
* Tag: 10000
* Usuario: 14000

Se harán tanto consultas como escrituras para observar distintas métricas como el porcentaje de error, los retardos y otra información importante.

**Hipótesis**:

SQLITE no permite modificaciones y lecturas simultáneas en la misma tabla. Por eso, se produce un “lockeo” cuando se agrega o modifica algún registro. Cualquier consulta que se quiera realizar en este momento, será rechazada. Cuando finaliza la modificación, la tabla puede consultarse nuevamente.

Debe tenerse en cuenta, que nuestra API, necesitará una buena respuesta frente a escrituras, ya que al finalizar un juego, este debe grabarse en el sistema, además, muchos usuarios intentarán subir canciones nuevas que no se encuentren en el sistema.

**Prueba**:

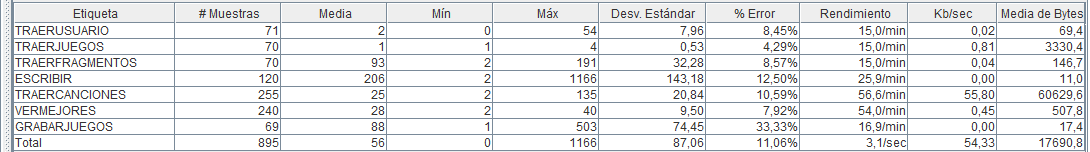
La prueba tratará de reflejar el siguiente caso, donde se observan 3 grupos:

* Grupo 1) 10 usuarios ingresan al sistema a lo largo de 5 segundos y realizan consultas sobre su usuario (perfil), ven sus juegos, piden un juego y luego lo suben.
* Grupo 2) Cada 15 segundos, ocurren picos de 15 usuarios que ingresan a observar canciones y usuarios.
* Grupo 3) Cada 7 segundos, un grupo de usuarios ingresa a escribir canciones en el sistema (se incluyen los respectivos fragmentos) en el sistema.

Cada acción realizada por los usuarios presenta una pequeña demora para tratar de representar de forma más realista el comportamiento del usuario

La prueba se repite indefinidamente (tratando de dar la apariencia que los usuarios no se desconectan sino que se mantienen).

A continuación se ingresan los resultados luego de varios minutos:



* TRAERUSUARIO, TRAERJUEGOS, TRAERFRAGMENTOS Y GRABARJUEGOS pertenecen al grupo 1.
* ESCRIBIR pertenece al grupo 3.
* TRAERCANCIONES y VERMEJORES pertenecen al grupo 2.

Puede observarse en la imagen que podemos mantener 3.1 operaciones/segundo con una tasa de error del 11.05%.

Los principales focos de error son las escrituras: GRABARJUEGOS y ESCRIBIR. Las cuales, como dijimos previamente, lockean sus tablas.

Además puede observarse que las consultas VERMEJORES, TRAERCANCIONES y TRAERFRAGMENTOS tienen un tiempo de ejecución bastante más alto que el de TRAERUSUARIO y TRAERJUEGOS. Traer fragmento es el más importante de estas consultas, ya que es la base del juego.

**Plan de mejora**:

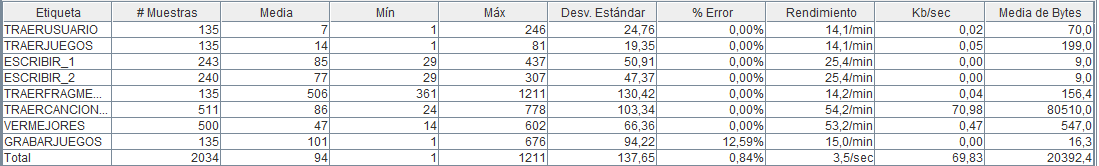
Si bien SQLite es una base de datos simple y útil para base de datos pequeñas, no es lo adecuada para esta clase de aplicación. Por lo que se propone pasar de SQLite a una base de datos que permita mejor concurrencia ante escrituras y lecturas simultaneas.

El objetivo es observar si este reemplazo genera un cambio drástico del rendimiento de las consultas y en consecuencia, el de la API. Además de ver si el error disminuye.

La base de datos a la cual portar será MariaDB.

**Aplicación de la mejora**:

Luego de portar la base de datos a MariaDB se realizaron las mismas pruebas que antes, dando como resultado:



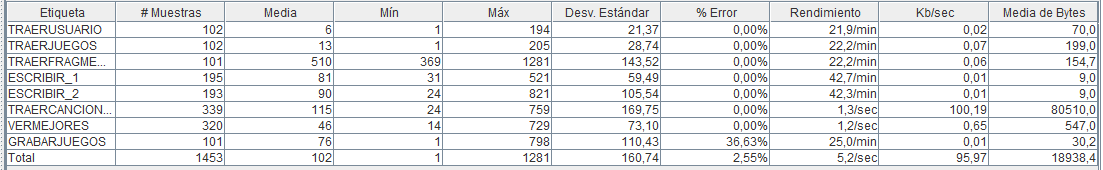
(Escribir fue separado en 2 requests distintos, debido a un error en la llamada desde Jmeter).

Como se puede observar, el error se redujo en casi un 10%, logrando que ahora podamos hacer 0.4 consultas correctas extra por segundo (24 consultas más por minuto).

[GRABARJUEGOS presenta un leve error, debido a un problema con el TIMESTAMP que no es reconocido correctamente para la clave primaria, suponemos, que de no ocurrir este problema, el error sería casi nulo]

Además, ahora que el motor de base de datos maneja mejor la concurrencia, las lecturas ya no son fuente de errores (100% éxito).

A continuación, se agrega una prueba con una base de usuarios mayor (casi un 50% más en cada grupo).

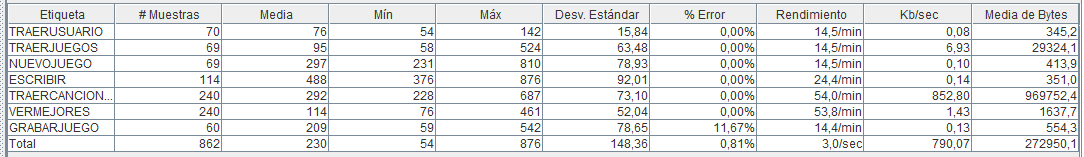


Al igual que antes, el error es despreciable.

**Conclusiones**:

El cambio desde un motor de base de datos no apto para aplicaciones masivas y concurrentes a uno más adecuado para estas situaciones permitió reducir considerablemente la tasa de error. De esta forma, mejoro la disponibilidad de nuestra API

**API con sqlite**



El error se redujo debido a que CodeIgniter implementa timeouts para la base de datos, por lo tanto si está lockeada, vuelve a intentarlo, sin embargo esto incrementa en gran medida el tiempo de respuesta a las peticiones.

Puede observarse por ejemplo los requests de TRAERUSUARIO y TRAERJUEGOS comparados con los mismos sin pasar por la API (aproximadamente 70 veces el tiempo de respuesta).

Además podemos ver como la media de bytes se incrementa considerablemente debido a que estas son peticiones HTTP y no consultas directas a la base de datos.

**API con mysql**