Arquitectura cliente-servidor concurrente para framework web (Septiembre 2019)

David Daniel A. Ramirez Moreno, Estudiante Ingeniería Sistemas

I. Introducción

Este documento presenta una aproximación a la implementación y el Desarrollo de un framework web que permite realizar distintas peticiones de manera concurrente, esto permite optimizar los resultados y el uso del sistema donde dicho framework se encuentre implementado y corriendo. Esto se debe a que por el hecho de ser concurrente le permite manejar distintas peticiones al mismo tiempo en lugar de manejarlas una por una.

El servidor esta implementado para devolver recursos html e imágenes .jpg que pueden ser agregadas en el respectivo directorio para su posterior solicitud desde la dirección web.

Adicionalmente el servidor permite la invocación remota de métodos para lo cual se pueden agregar los diferentes POJOs e implementarlos para su llamado.

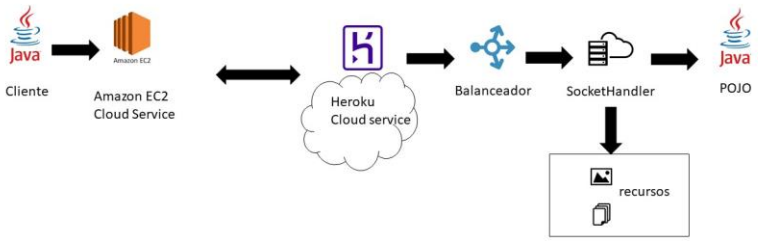
La idea detrás de este archivo es realizar una prueba de estrés sobre el servidor, haciendo múltiples peticiones al mismo tiempo con un cliente que también es concurrente, a dichas peticiones se les monitoreara el tiempo, se sacara un promedio y se realizaran gráficos que permitan analizar el rendimiento del servidor.

El servidor se encontrará en heroku mientras que el cliente será desplegado en AWS.

se mostrará la arquitectura y el diagrama de despliegue tanto del aplicativo como del cliente para entender su funcionamiento y el desarrollo de las estadísticas.

Por ultimo se tomarán las estadísticas y se realizara un análisis de estas, teniendo en cuenta el tiempo de respuesta y el número de solicitudes realizadas.

II. Arquitectura de Software



En el grafico anterior se evidencia la arquitectura presentada en este proyecto, dado que es una arquitectura para una prueba de estrés al servidor, se plantearon 2 máquinas que están alojadas en la nube.

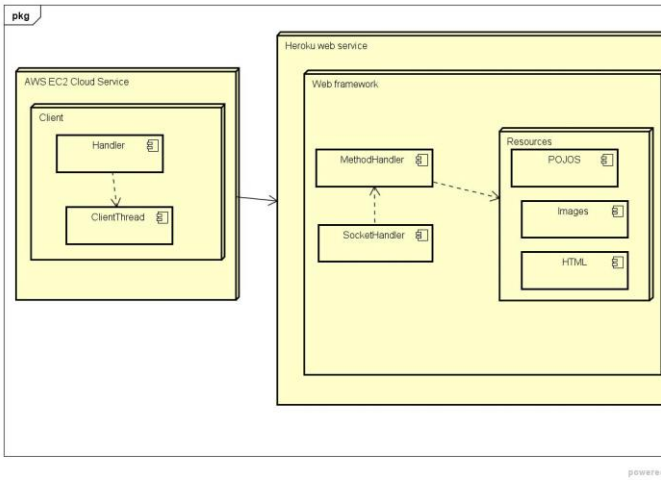
La primera representa la maquina del cliente, el cliente consta de 2 clases que se encuentran desplegadas en AWS EC2 la labor principal del cliente es realizar una cantidad determinada de peticiones al servidor y calcular el tiempo que se demora el servidor en responder dichas solicitudes.

La segunda maquina representa el servidor, consta de la implementación recurrente del framework realizado en el artículo anterior.

Esta maquina se encuentra desplegada en heroku, se encarga de manejar las distintas peticiones de manera concurrente, para esto cuenta con un balanceador o executor, la labor de este es recibir las peticiones, generar un hilo que la procese y poner al servidor a escuchar una nueva petición.

Dicho hilo es representado en el diagrama como el socketHandler, aquí evalúa que tipo de recurso le están solicitando y en base a esto lo busca y lo retorna en caso de existir.

III. Deployment Diagram



El despliegue consiste en 2 componentes principales, el primero es el servidor desplegado en heroku y el segundo es el cliente desplegado en AWS,

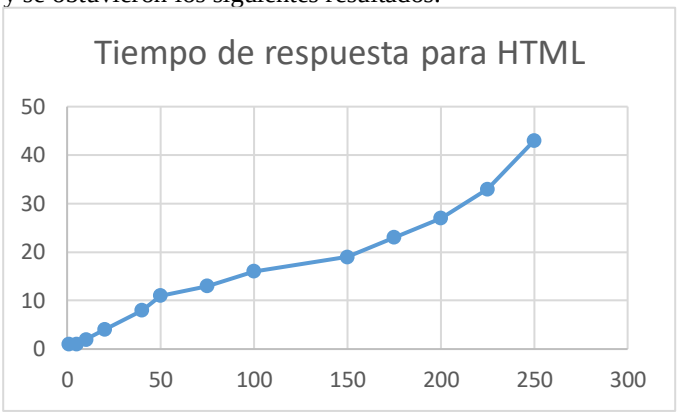
En el servidor de heroku contamos con 2 clases principales, la primera es el socket Handler, esta clase contiene el pool de hilos y es la que gestiona cuantas peticiones esta manejando el servidor, la segunda clase importante es el method handler, esta clase recibe el socket del cliente con la petición ya realizada, lee dicha petición y la procesa respondiendo en caso de ser un html, una imagen o un método definido en el POJO, esta clase es la llamada MethodHandler y en si es un hilo que termina su ejecución al terminar de procesar la petición.
El handler se encuentra pendiente de los hilos que ha creado, cuando terminan su ejecución lo remueve del pool para poder responder a otras solicitudes que puedan llegar al servidor

En el servidor de AWS se cuenta con una implementación que simula varios clientes con varias peticiones, consiste de 2 clases, la primera es el Handler, esta clase recibe la url a la cual se le realizaran las peticiones y el número de clientes que se desean crear, por cada cliente que se indique, el handler creará un hilo ClientThread.
El hilo ClientThread es un hilo que recibe una URL, a dicha url le realizará 20 peticiones y medirá el tiempo de respuesta de las mismas, posteriormente un promedio de dichos tiempos de respuesta será realizado y mostrado en pantalla.

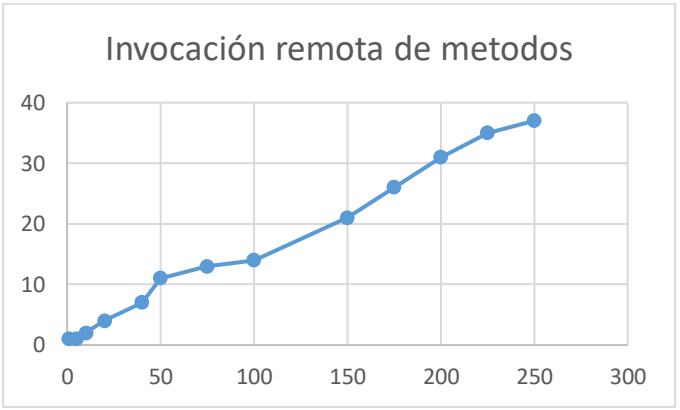
IV. Pruebas

La parte de pruebas será distribuida en 3 partes, una para cada tipo de respuesta que se puede obtener del servidor, es decir imágenes, html e invocación remota de métodos.
Las graficas tienen en el eje X el numero de hilos de clientes, cada cliente realiza 20 peticiones y promedia el tiempo que se demora el servidor en responder cada petición, posteriormente dicho tiempo es mostrado en el eje Y.

1. Para la primera prueba se solicitó un recurso HTML y se obtuvieron los siguientes resultados:



2. Para la segunda prueba se invocaron métodos remotamente y se obtuvieron los siguientes resultados:



3. Para la ultima prueba se solicitaron imágenes y se obtuvieron los siguientes resultados:

Como se observa en las gráficas, el tiempo de respuesta para los html planos y los métodos remotos es muy similar, sin embargo las imágenes se demoran mucho mas en ser respondidas, adicionalmente se evidencia que entre más solicitudes concurrentes son realizadas al servidor le cuesta más responderlas, disminuyendo el tiempo de respuesta de manera significativa a mayor número de solicitudes.

V. Conclusion

Un servidor no concurrente es bueno para cuando se realizaran pocas peticiones, sin embargo, si se desea sacar el máximo potencial de un servidor y equipo lo mejor es implementarlo de forma concurrente, esto permite una mayor velocidad, agilidad y tiempos de respuesta a los clientes lo que se traduce en una mejor relación con los mismos, además de esto muchos usuarios pueden acceder al tiempo.

Se aprendió a usar AWS con su servicio EC2, en el mismo se logro montar un cliente desarrollado en java y ejecutarlo, lo que genera conocimientos futuros a la hora de usar un servicio en la nube de Amazon.

Las estadísticas nos permiten evaluar el rendimiento de un aplicativo cuando se le somete a pruebas de estrés, lo cual permite evaluar que practicas son mejores y que se puede realizar.