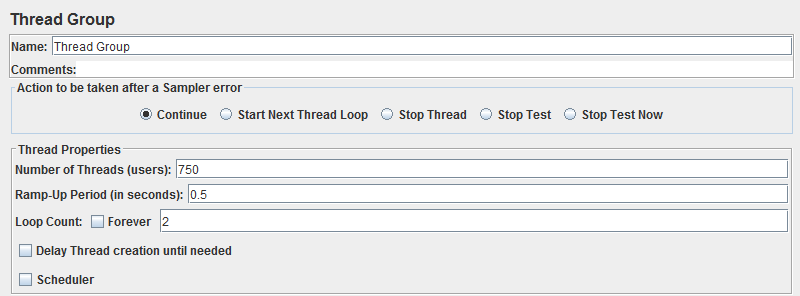
Para el desarrollo del sistema se usó una arquitectura multi-tier, esto con el fin de favorecer el desempeño y escalabilidad del sistema; en concreto usamos una arquitectura de dos tiers donde tenemos una tier de lógica (Backend) y otro donde se realiza la persistencia del sistema (Base de datos). Estas decisiones se hicieron basadas reflexionando en la distribución de cargas a través de los tiers, con el fin de mejorar los tiempos de ejecución del sistema aprovechando los recursos disponibles en cada uno de ellos, sumado a esto, se pensó en la escalabilidad del sistema ya que gracias a la arquitectura escogida los nuevos servicios que se quieran agregar podrán ser fácilmente adicionados sin alterar la lógica de los actuales.

De igual modo, hicimos uso de Jax RS para la realización del sistema con el fin de tener un completo control sobre los servicios REST; dado que, Jax-RS nos permite que la implementación del servicio REST que será consumido por el cliente sea sencilla de hacer gracias a las anotaciones implementadas dentro de java, además de permitir la escalabilidad del servidor sin afectar los servicios del REST, esto debido a que Jax-RS se encarga de “consumir” o generar los HTTP request y traducirlos para que el servidor no tenga que entender la lógica de los mismos.

Para las pruebas se usó una configuración de Jmeter en cluster. Desde 2 computadores diferentes se coordinaban pruebas como se muestran a continuación:

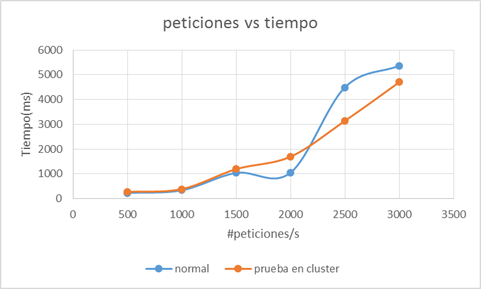


Se usaron 750 threads que se disparaban en medio segundo con una iteración. Esto hacia que cada uno de los threads creados en cada uno de los 2 computadores fuera reutilizado, para tener un total de 1500 peticiones en un segundo por cada computador. Al haber 2 computadores en el cluster, en total se probaba el programa con un total de 3000 peticiones por segundo.

Las pruebas en cluster permiten que no se gasten innecesariamente recursos de cada computador y hacen que el GUI de Jmetter no se sobrecargue.

Inicialmente se hizo un cargue de pacientes y luego se hicieron peticiones GET sobre estos pacientes.

También se hicieron las con una única máquina que enviaba las peticiones y se obtuvieron los siguientes resultados:

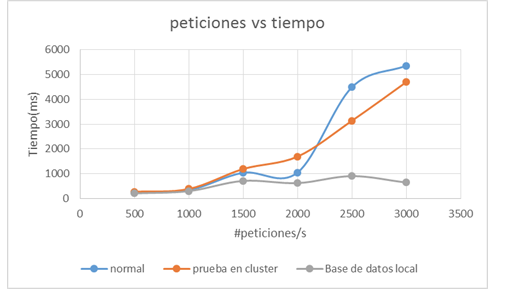


**Gráfica 2. GET pacientes**

Como puede observarse en la imagen, el tiempo de respuesta es muy similar para ambos tipos de pruebas, ya que estos dependen es del servidor.

Los tiempos de respuesta para las 3000 peticiones llega a ser casi de 5 segundos, un tiempo bastante alto. Este tiempo puede deberse al computador en dónde estaba corriendo el backend, ya que no es propiamente un servidor, además, también se debe tener en cuenta el tiempo de respuesta de la base de datos, ya que no se encuentra ubicada en la misma ubicación.

Usando la misma configuración de la prueba pasada se adicionó una base de datos local para comparar los tiempos de respuesta y la información se ve reflejada en la gráfica 3. El tiempo de respuesta ahora para las 3000 peticiones no sobre pasa el segundo usando una bsae de datos local. Esto puede ser cierto debido a que la base de datos está alojada en la misma ubicación dónde se corre el backend.



**Gráfica 3. GET pacientes**

Además, para la mejor configuración se probó POST de los 3000 simultáneos obteniendo esto:

