**Pre-Experimentación**

1. Problemática

Se busca desarrollar una aplicación que sea capaz de almacenar y manejar los datos de 1000 usuarios mensualmente con un buen desempeño, permitiendo la respuesta inmediata ante el manejo de distintas emergencias que se pueden dar por irregularidades en las condiciones médicas del paciente; dando la posibilidad de enviar un mensaje como respuesta al paciente o en un caso grave notificar a las diferentes entidades involucradas. Para solucionar esto, usaremos la arquitectura de tipo reactivo basada en un modelo de actores.

1. Objetivo

Soportar grandes cantidades de mensajes simultáneos, permitir el sistema de notificación y persistencia de los datos históricos de un paciente. También se busca que el software permita la manipulación de dispositivos externos.

1. Descripción

Se probará el programa con la recepción de 3000 sensores de diferentes tipos en una ventana de tiempo de 1 segundo. Se buscará obtener el historial de datos personales y otros datos relacionados con la historia clínica dentro de un rango de tiempos permitiendo el uso de los servicios CRUD. Se probará el envío de mensajes al dispositivo.

1. Artefactos a Construir

Se desarrollará una aplicación utilizando el framework “Play!” para el manejo de peticiones al programa y la arquitectura se implementará soportada en Akka para gestionar los threads y la reutilización de recursos evitando errores como “StackOverFlow” y “OutOfMemoryError”. Todo esto con el objetivo de solucionar el problema mencionado anteriormente.

1. Recursos de Experimentación

Para desarrollar el software se usarán una máquina virtual que cuenta con 8GB de RAM, un procesador Intel(R) Xeon(R) CPU a 2,6 GHz con sistema operativo Windows 7 de 64bits. También se utilizará el software IntelliJ como ambiente de desarrollo, Jmeter para realizar las diferentes pruebas de carga y desempeño, PostgreSQL para el manejo de bases de datos, Git-Hub para el modelo de trabajo de integración continua y Post-Man para probar el correcto funcionamiento del software.

1. Resultados Esperados

Se espera que la aplicación cumpla con los parámetros de escalabilidad y eficiencia para adaptarse a los cambios y al gran volumen de peticiones que llegan sin perder desempeño, tomando en cuenta las delicadas situaciones que se manejan. Se espera desarrollar una aplicación amigable con el usuario, es decir, hacer un sistema usable. También que sea permita la interoperabilidad.

1. Duración y Etapas

La planeación se dividió en 5 etapas con sus debidos tiempos esperados:

* Modelaje del mundo del problema: 1 Hora
* Determinar requerimientos funcionales y no-funcionales: 30 Min
* Identificar las características del problema: 30 Min
* Implementación del software: 24 Horas
* Pruebas: 1.5 Horas

Escenarios de Calidad

Para mejorar los escenarios de calidad nos ayudó el framework play pues este favorece la escalabilidad y la concurrencia, mediante el uso de componentes livianos y sin estado. Del mismo modo, al recibir las solicitudes de los clientes, las desliga y las parte en subtareas, para procesarlas con múltiples actores de akka de manera asincrónica y no bloqueante. También usa la característica de java FutureTask que permite pausar peticiones y liberar el threat mientras se culminan otros procesos para continuar con la primera petición.

**Post-Experimentación**

1. Resultados Obtenidos
   1. Primera Prueba

Después de hacer las pruebas en Jmeter tomando 1000 muestras para 3 sensores se obtuvieron los siguientes resultados.

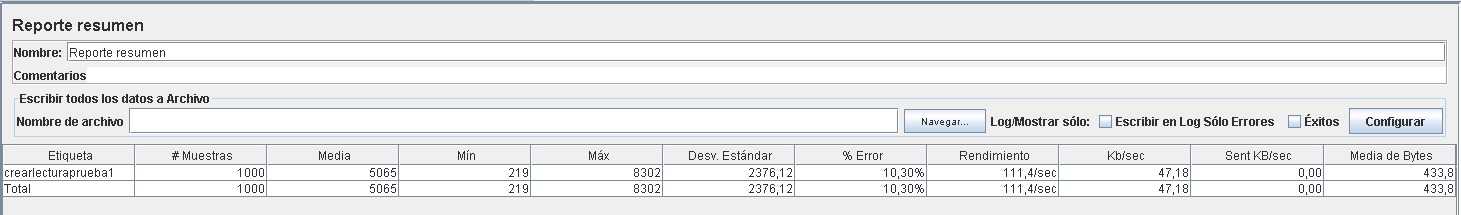


Imagen 1.

En la imagen 1 se puede ver que el porcentaje de error es del 10,30%, lo cual es un porcentaje bajo ya que la cantidad de muestras es muy elevada y son procesadas rápidamente. Éstos errores se pueden presentar por el gran número de solicitudes que se generan, las cuales sobrecargan al sistema ya que en algún momento puede que la conexión con la base de datos no sea tan rápida como se espera.

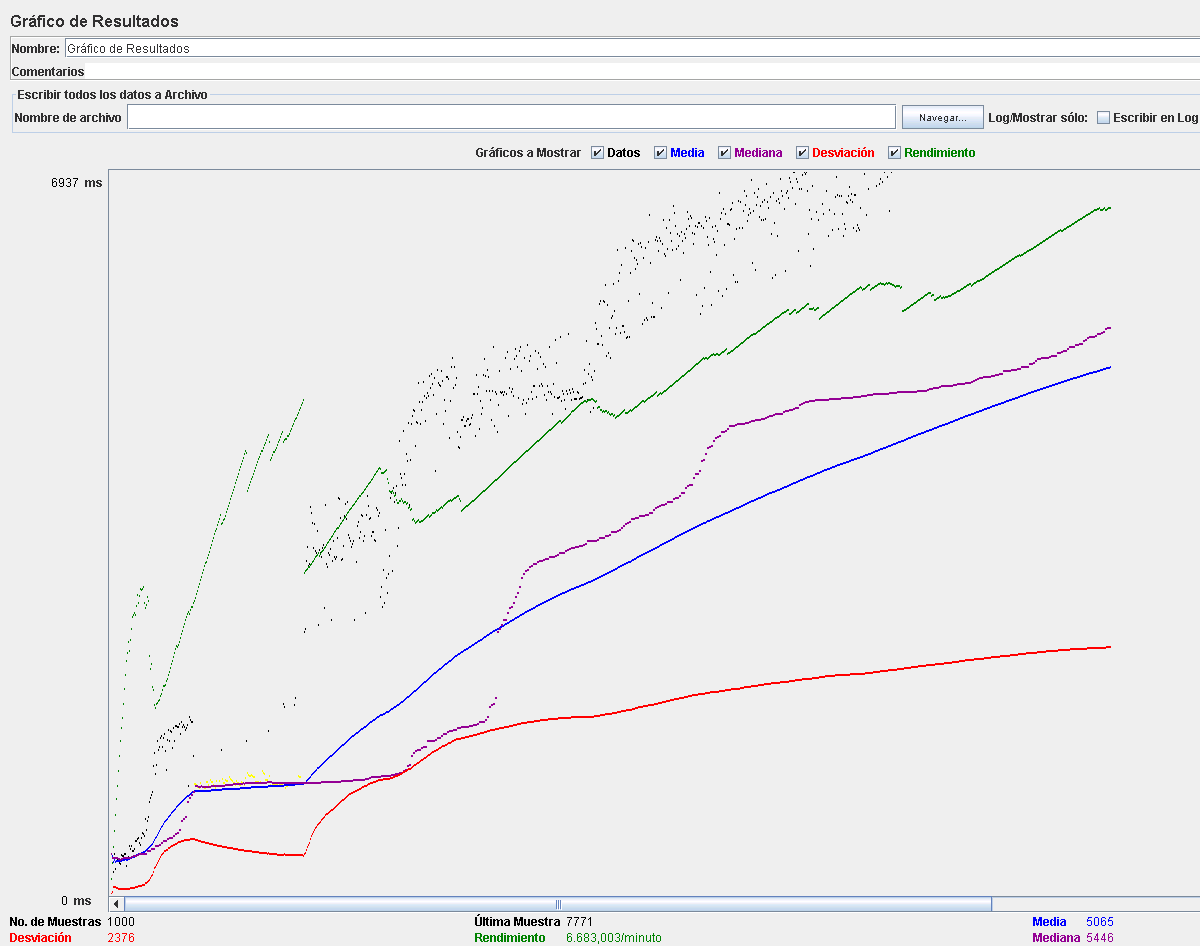


Imagen 2.

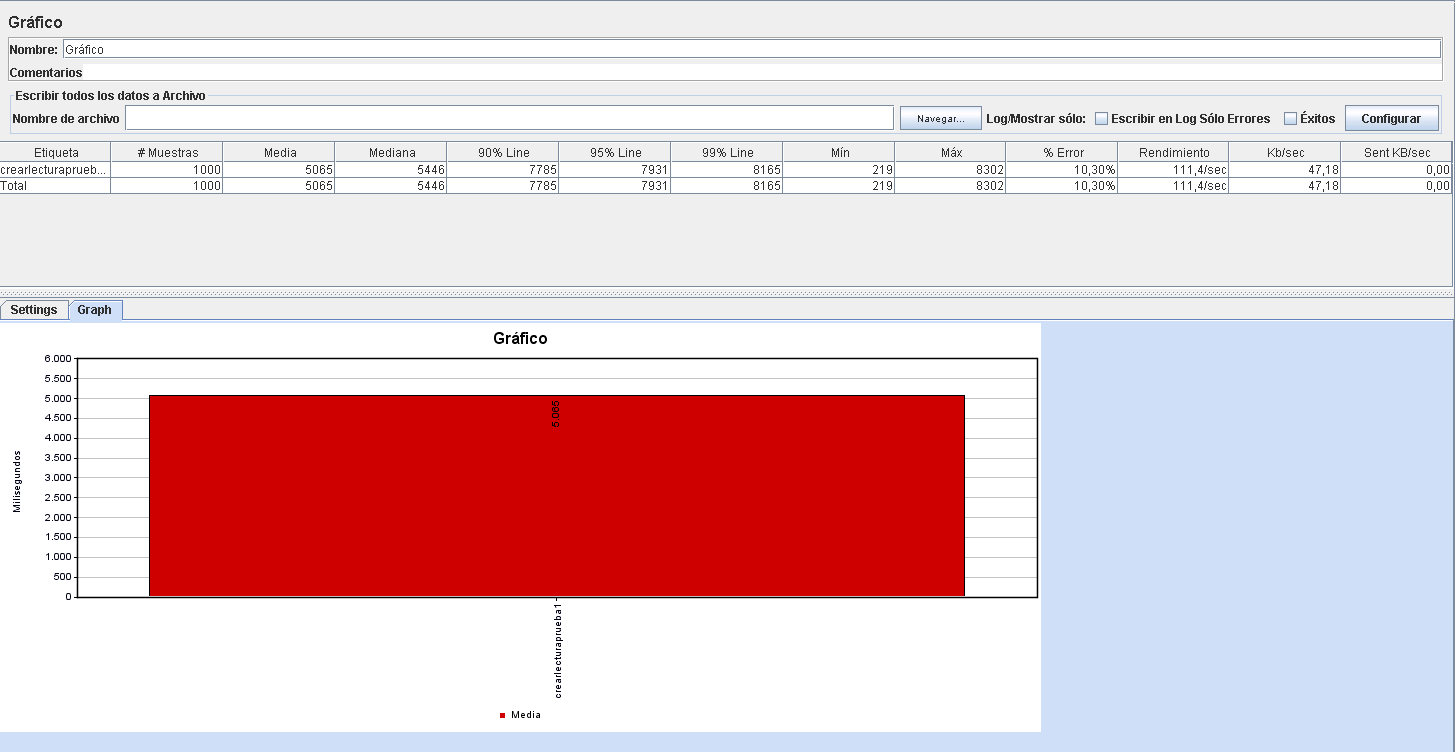


Imagen 3.

En las imágenes 2 y 3 se ve que a medida que el número de muestras va incrementando, el tiempo de respuesta también aumenta, pero no de una forma exponencial. Por lo tanto, aunque no cumple exactamente el atributo de calidad de escalabilidad no se aleja mucho del tiempo requerido. Todo esto teniendo en cuenta que la media es 5 segundos y la desviación es de 2,3 por lo que se puede decir que cada solicitud es atendida en un tiempo de 3 a 7 segundos como máximo. Lo cual puede ser causado por una sobrecarga, causada por una conexión lenta con la base de datos.

* 1. Segunda prueba

Después de los cambios, se realizaron nuevamente las pruebas en Jmeter tomando 1000 muestras para 3 sensores se obtuvieron los siguientes resultados.

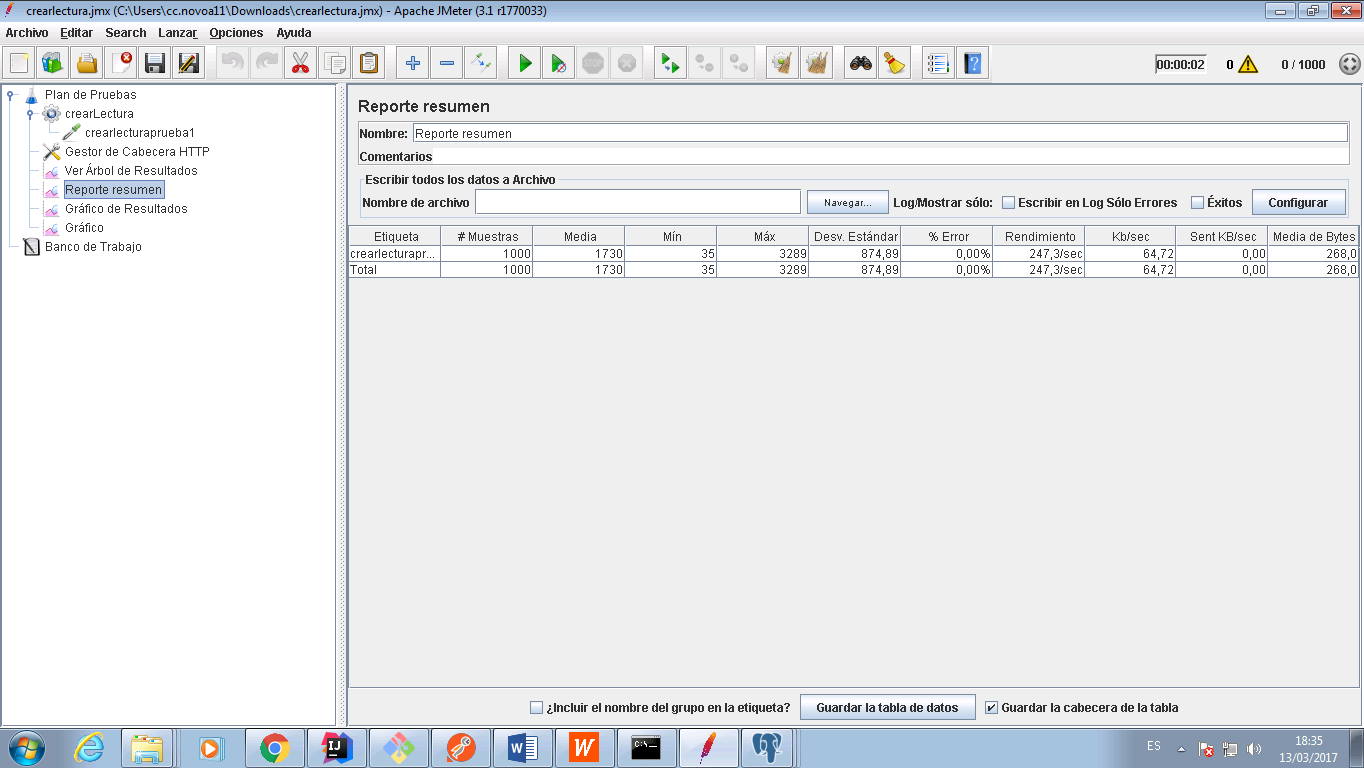


Imagen 4.

En la imagen 4 se puede ver que el porcentaje de error es del 0,0%, como se puede observar hubo una gran mejoría en la eficiencia, esto debido al uso de mayor cantidad de treads en la ejecución que permitieran la recepción más rápida y segura de solicitudes que llegan al programa.

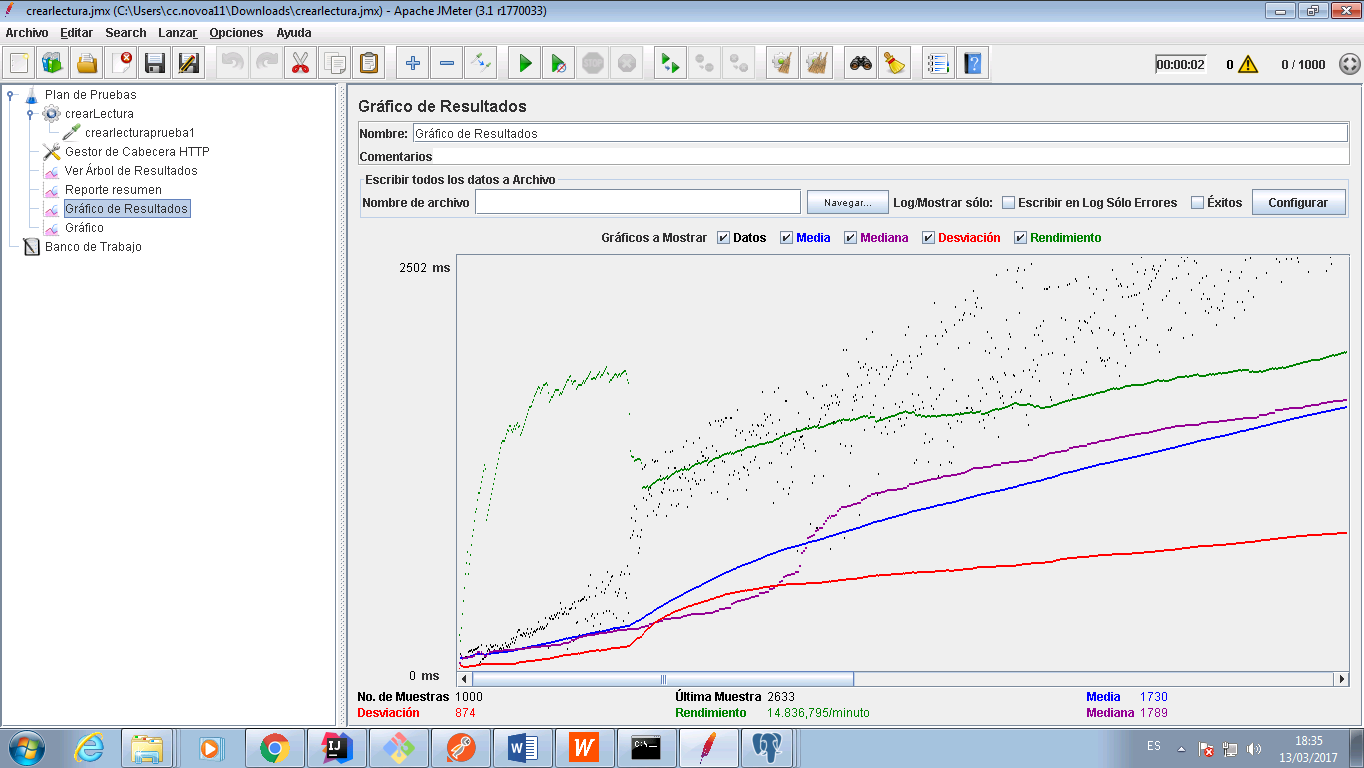


Imagen 5.

En la imagen 5 se ve que el programa mejoro en todo aspecto por un factor de 3 permitiendo observar lo importante que es la cantidad de treads que se disponen a la hora de ejecución de un programa. Con estos resultados la media es 1.7 segundos y la desviación es de 0.874 por lo que se puede decir que cada solicitud es atendida en un tiempo de 0.9 a 2.6 segundos como máximo.

1. Duración Real

* Modelaje del mundo del problema: 1.5 Horas
* Determinar requerimientos funcionales y no-funcionales: 30 Min
* Identificar características del problema: 30 Min
* Implementación del software: 18 Horas
* Prueba: 40 Min

1. Artefactos Construidos

Se desarrollaron todos los artefactos planteados en la pre-experimentación usando las herramientas mencionadas anteriormente. Obteniendo resultados satisfactorios respecto a los objetivos planteados.

Además se usó un controlador para simular el envió de datos, se usó una tarjeta wiring, cargada con el siguiente código:



Imagen 6.

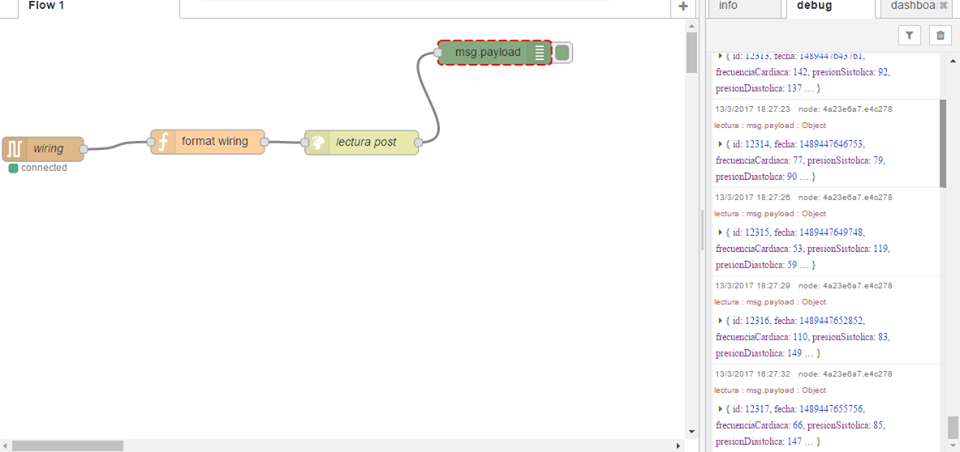
El código genera 3 valores aleatorios que corresponden a los tres valores de frecuencia cardíaca, presión sanguínea y nivel de estrés. Estos datos son consumidos por el programa mediante el siguiente modelo en node red:  


Imagen 7.

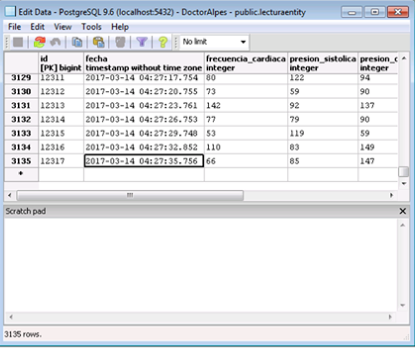


Imagen 8.

Como se ve en la imagen 7 se lee el formato de datos que arroja la wiring y después se pasan por un servicio post para cargarlos a la base de datos. Y como se ve a la derecha y en la imagen 8 los datos se persisten satisfactoriamente.

1. Análisis

Como se puede observar en la gráfica 2, el orden de procesamiento no es lineal debido al sistema de FutureTask. Adicionalmente vemos que el error no es nulo y es muy alto para el tipo de servicio que se presta, pero esto se puede ir mejorando aplicando mejorías al modelo de arquitectura aplicado. Vemos que aun así se lograron las metas esperada para el rápido procesamiento de grandes volúmenes de datos.

1. Conclusión

* La arquitectura planteada funciona correctamente ya que permite un correcto manejo de peticiones al programa y una correcta gestión de threads, teniendo en cuenta el gran volumen de peticiones que se van a generar. Aunque falta mejorar la parte de la conexión con la base de datos, la cual se cree que es el problema para que no se presenten fallas en la parte de escalabilidad.
* La arquitectura planteada funciona correctamente, aunque falta mejorar la parte de la conexión con la base de datos para que no se presenten fallas en la parte de escalabilidad.