Sistema de Seguimiento Cardiaco

Juan Sebastián Barragán Jerónimo, Sebastián Sánchez Galiano, Santiago Rojas Herrera

Experimento 1 – Entrega definitiva

{s.rojas19, js.barragan796, s.sanchez59}@uniandes.edu.co

Fecha de presentación: 13 de marzo de 2017

Bogotá, Colombia

Tabla de contenidos

1. Pre-experimentación

1.1. Problemática

Se debe construir un sistema para el Hospital Cardiológico de Santa Fé que permita la administración y acción a partir de la información de pacientes y médicos, con funciones como:

- Ingreso de mediciones cardiacas dadas periódicamente por un brazalete.
- Envío de consejos por parte de un médico a un paciente.
- Actualización y revisión de exámenes, diagnósticos y tratamientos de un paciente.
- Recibimiento de emergencias de un paciente y acción sobre estas.
- Reconfiguración del marcapasos de un paciente.
- Entre otros…

1.2. Objetivo del experimento

Se implementará este sistema sobre una arquitectura de capas, que hará uso de una base de datos relacional; Adicionalmente, este sistema será ejecutado sobre un servidor único que albergará servicios, lógica y persistencia considerando que una implementación más compleja está fuera del alcance de este experimento y de limitaciones de tiempo.

1.3. Descripción del experimento

En el documento se expondrán pruebas de carga para el envío de mediciones y emergencias desde un dispositivo externo hacia el servidor de la aplicación. Para esto, se hará uso de la herramienta *JMeter*, que permite simular el envío de múltiples peticiones (en este caso http) hacia la aplicación.

Adicionalmente se implementarán pruebas para las funciones en *Postman*, aunque sus resultados no serán documentados en este documento, pues serán demostrados al momento de sustentar.

1.4. Artefactos a construir

Como se vió en el documento de arquitectura de software, la división de los componentes generalmente se da por:

- Consejos
- Emergencias
- Historia Clinica
- Marcapasos (tratamientos, diagnósticos y exámenes)
- Mediciones
- Médicos
- Pacientes
- Reservas

Dicho esto, se construirá:

- Interfaz de servicios y lógica
- Entidades
- Persistencia
- DTOs
- Recursos

Adicionalmente, se implementará la configuración de los sensores en Node-red, que permite el envío de mediciones al servidor de aplicación por medio de http–REST

1.5. Recursos de la experimentación

- Macbook Pro 15 (2015)
    - Core i7 2.2GHz, 4 núcleos
    - 16GB 1600MHz Ram
- Macbook Pro 13 (2014)
    - Core i5 2.7GHz, 2 núcleos
    - 8GB 1866MHz Ram

1.6. Resultados esperados

Se espera que todos los componentes del software sean funcionales, y que el sistema soporte la recepción de información de 3000 sensores (3 sensores x 1000 pacientes) en una ventana de tiempo de 1 segundo, igualmente con emergencias.

1.7. Duración y etapas

Ya se desarrolló la entrega preliminar de este experimento. En esta, se implementó la gran mayoría de los servicios de la aplicación principal (Lógica, entidades, persistencia, DTOs y recursos), aunque varios de los servicios necesitaban de mejoras.

Así, solo resta:

Actividad	**Tiempo de desarrollo**
Corregir y mejorar la aplicación principal	~6 horas
Implementar aplicación en node-red para el microcontrolador	~6 horas
Implementar y probar pruebas de la aplicación	~3 horas

2. Experimentación

2.1. Recepción de información de los sensores

Se realizaron unas pruebas de carga sobre la aplicación. En estas pruebas se comprobó que el sistema soporta la recepción de información desde los 3.000 sensores (3 sensores x 1.000 pacientes potenciales) en una ventana de tiempo de un segundo. Para la configuración de las pruebas se utilizaron solo 1.000 threads considerando que una decisión de arquitectura fue utilizar una solo solicitud para enviar la información de los tres atributos: frecuencia cardiaca, presión sanguínea y nivel de estrés. Está decisión es explicada en la sección de arquitectura de este documento.

Entrega Parcial

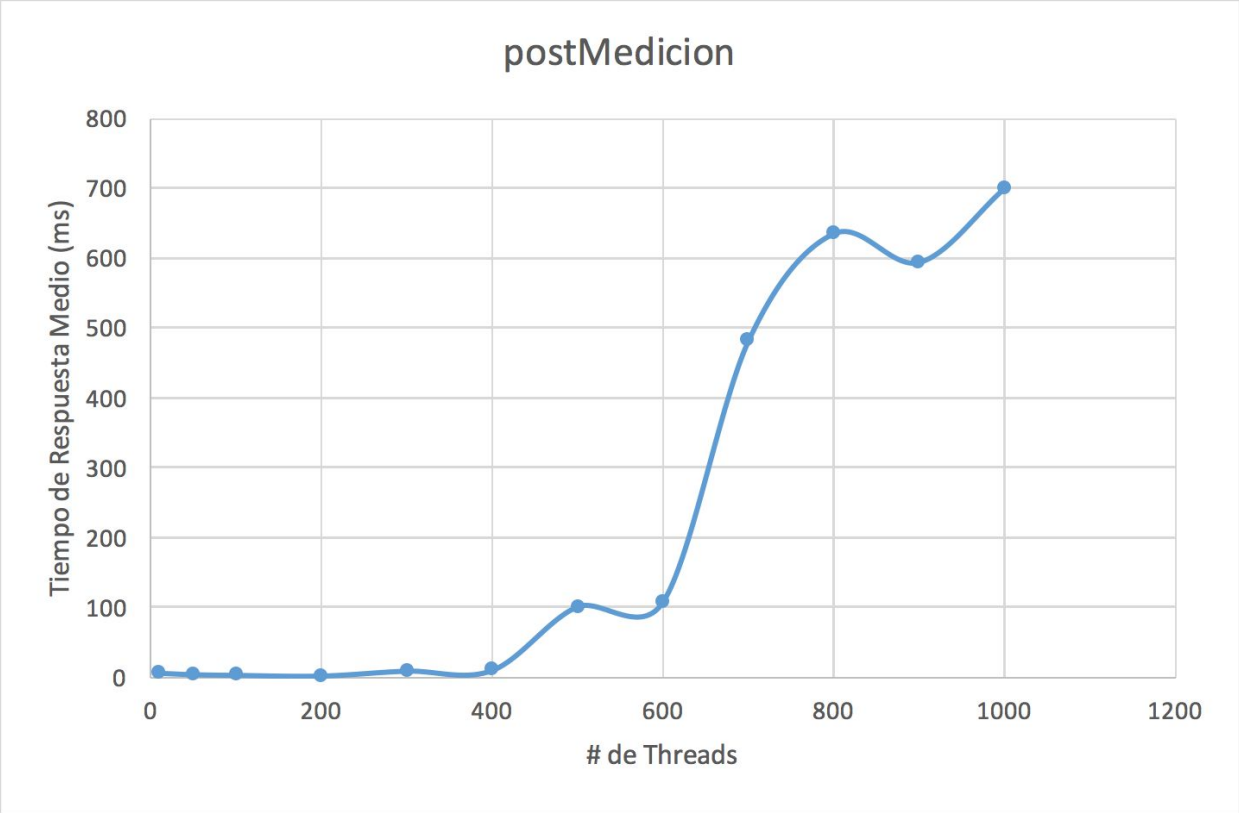
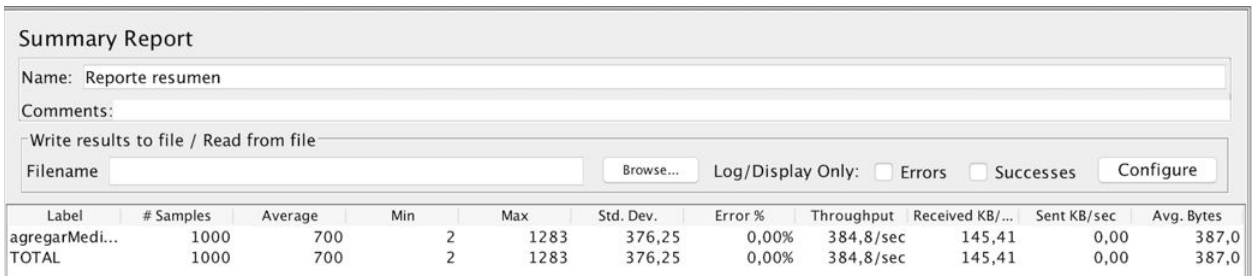


Figure 1. Tiempo de Respuesta Medio para Crear una Medición (Entrega Parcial)

En la gráfica anterior se puede comparar el tiempo de respuesta medio dependiendo del número de threads lanzados. Se puede ver que para números de threads menores e iguales a 1.000 el tiempo de respuesta medio es menor a 1 segundo. También se puede ver que el tiempo de respuesta medio aumenta drásticamente cuando hay más de 600 threads. El tiempo de respuesta medio para 1.000 threads es 700ms.

A continuación, se muestra el pantallazo de JMeter para la prueba realizada con 1.000 threads.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
agregarMedi...	1000	700	2	1283	376,25	0,00%	384,8/sec	145,41	0,00	387,0
TOTAL	1000	700	2	1283	376,25	0,00%	384,8/sec	145,41	0,00	387,0

Figure 2. Reporte Resumen de JMeter (Entrega Parcial)

Entrega definitiva

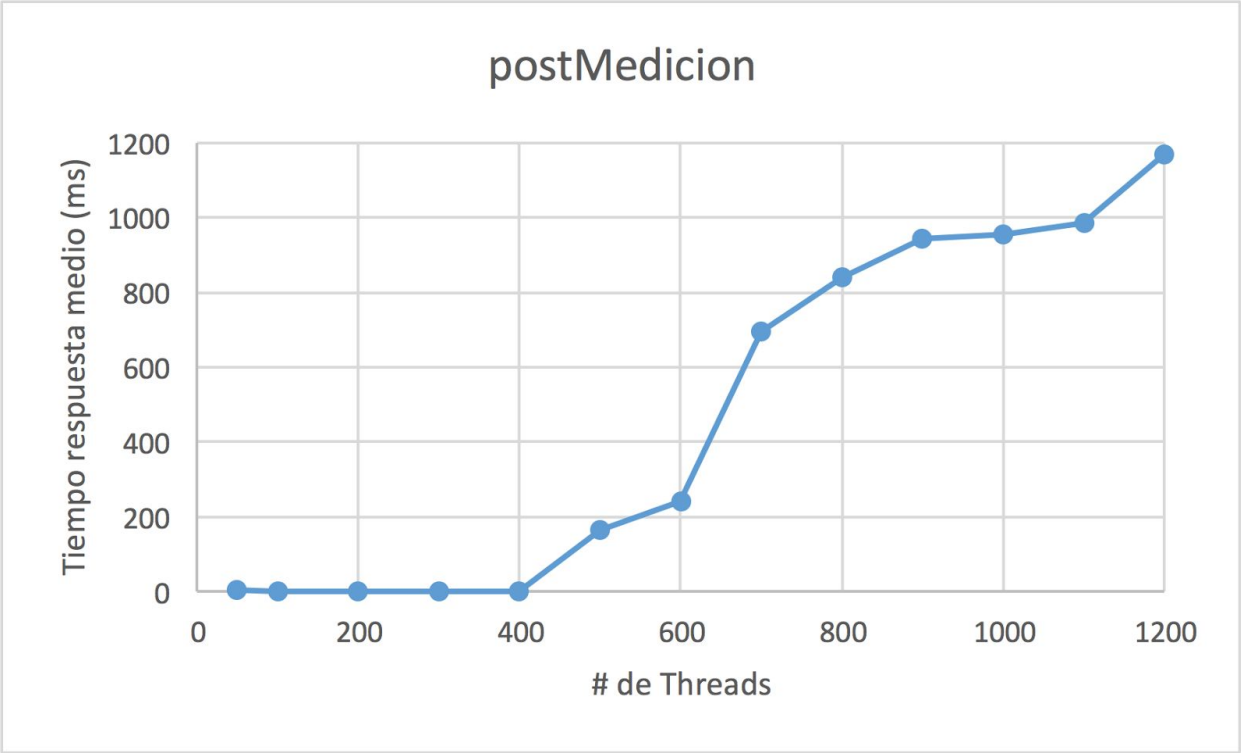


Figure 3. Tiempo de Respuesta Medio para Crear una Medición (Entrega Definitiva)

En la gráfica anterior se puede comparar el tiempo de respuesta medio dependiendo del número de threads lanzados. Se puede ver que para números de threads menores e iguales a 1.000 el tiempo de respuesta medio es menor a 1 segundo. También se puede ver que a medida que aumenta el número de threads aumenta el tiempo de respuesta medio. El tiempo de respuesta medio para 1.000 threads es 985 ms.

A continuación, se muestra el pantallazo de JMeter para la prueba realizada con 1.000 threads.

Figure 4. Reporte Resumen de JMeter (Entrega Definitiva)

Se puede hacer una comparación de los resultados obtenidos en la entrega parcial y en la entrega definitiva. En primer lugar, cabe notar que el tiempo de respuesta medio con 1.000 threads para la entrega definitiva es mayor al tiempo encontrado en la entrega parcial. Los valores respectivos fueron 985 ms y 700 ms. Esto se puede deber a los cambios que se hicieron en la clase medición. Se hicieron modificaciones sobre esta clase para que incluya una emergencia si los valores de los signos vitales registrados en la medición sobrepasan algunos límites normales. Aparte de lo anterior, se puede ver que los curvas de número de threads vs tiempo de respuesta medio para ambas entregas se comportan de forma muy similar. En ambas curvas se puede ver que el tiempo de respuesta medio aumenta drásticamente cuando hay más de 600 threads.

2.2. Recepción de emergencia

Para cumplir con el requerimiento de post emergencia se realizó una modificación en la clase medición. Esta modificación permite que se le adicione una emergencia a la medición si la lógica del lado de NODE-RED detecta que los valores recibidos por los sensores se encuentran por fuera de los rangos normales. Cabe destacar que las pruebas de medición para la entrega definitiva se realizaron con mediciones que contienen emergencias. Por lo anterior, las pruebas de recepción de 1.000 mediciones deben ser tenidas en cuenta también como pruebas de recepción de 1.000 emergencias.

3. Post-experimentación

3.1. Duración real

Actividad	**Tiempo de desarrollo**	Δ
Corregir y mejorar la aplicación principal	~6.5 horas	~0.5 horas
Implementar aplicación en	~7.5 horas	~1.5 horas

node-red para el microcontrolador		
Realizar pruebas de la aplicación	~3 horas	~0 horas

3.2. Artefactos construidos

Todos los artefactos propuestos fueron construidos.

Adicionalmente, se implementaron funcionalidades en el microcontrolador que permiten el envío de información de sensores reales.

3.3. Análisis – Arquitectura

Para favorecer el cumplimiento de los escenarios de calidad se tomaron unas decisiones en cuanto a la arquitectura a implementar.

En primer lugar, se tomó la decisión de implementar una arquitectura por capas, más específicamente se eligió implementar una arquitectura JAX-RS. Implementar una arquitectura por capas permite que haya bajo acoplamiento y alta cohesión. El hecho de que la solución tenga estas características permite que el software tenga robustez, confiabilidad, reusabilidad y facilita su entendimiento. La justificación para utilizar una arquitectura de capas es que se realizó un experimento en el que se comparó la arquitectura JAX-RS con la arquitectura Play y se encontró que JAX-RS es mejor que Play en lo que se refiere al atributo de calidad de escalabilidad. JAX-RS pudo manejar mejor el crecimiento continuo de trabajo de manera fluida y pudo hacerse más grande sin perder calidad en los servicios ofrecidos. La elección de una arquitectura por capaz es justificada teniendo en cuenta que se quiere ofrecer una solución de software que atienda a un número de usuarios considerable.

En segundo lugar, se tomó la decisión de implementar una solución mediante servicios web RESTful. Implementar esta arquitectura permite cumplir con los escenarios de calidad para el atributo de desempeño. Esto se debe a que hay un protocolo cliente/servidor sin estado y como resultado ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Esto se traduce en un mejoramiento en las métricas de latencia y un bajo uso de los recursos computacionales. Otra decisión que se tomó fue implementar beans sin estado. La

justificación para hacer esto es similar a la anterior y se traduce a que la aplicación tiene un mejor desempeño.

Otra decisión que se tomó está relacionada a la manera en que el sistema recibe la información de los sensores de frecuencia cardiaca, presión sanguínea y nivel de estrés. Se eligió utilizar solo una petición para mandar la información de estos tres atributos. Si se mandará la información de cada uno esto supondría un aumento en el overhead computacional lo que no favorecería el desempeño de la aplicación.

Finalmente, por restricciones de tiempo se decidió ejecutar la aplicación en un único servidor, viendo que hacer una solución que haga uso de balanceadores de carga resulta en un aumento considerable del tiempo de desarrollo, y que con lo implementado en el laboratorio de configuración de un balanceador de carga se lograron peores tiempos a los dados en una solución con un solo servidor (probablemente por mejoras necesarias), contrario a lo que la teoría dicta.

3.4. Conclusiones

En conclusión, la solución implementada fue suficiente para lograr los objetivos establecidos en el experimento, aunque es altamente probable que no sea la mejor solución. Es posible que el uso de una base de datos no relacional, como Cassandra, hubiese mejorado el rendimiento de la aplicación en ciertas instancias, al igual que el uso de balanceadores de carga y varios nodos con la aplicación, aunque estas alternativas hubieran aumentado considerablemente los tiempos de desarrollo.