**DOCUMENTACIÓN DE PROYECTO**

**SISTEMA DE ALERTA TEMPRANA DE TSUNAMIS (SATT)**

ISIS 2503 – Arquitectura y Diseño de Software

2016-10

Juan Santiago Acevedo – [js.acevedo10@uniandes.edu.co](mailto:js.acevedo10@uniandes.edu.co)

Alberto Mario Consuegra – [am.consuegra10@uniandes.edu.co](mailto:am.consuegra10@uniandes.edu.co)

Daniel Soto Rey – [d.soto11@uniandes.edu.co](mailto:d.soto11@uniandes.edu.co)

Mariana Villamizar Rodríguez – [m.villamizar564@uniandes.edu.co](mailto:m.villamizar564@uniandes.edu.co)

**I. Descripción del proyecto**

El Sistema Nacional de Detección de Alerta de Tsunamis se propone desarrollar un Sistema de Alerta Temprana de Tsunamis para determinar el nivel de peligro, establecer las zonas de evacuación y asignación de recursos de emergencia/rescate ante la presencia de un tsunami. Este sistema debe ser preciso, efectivo y tener una adecuada comprensión de los procesos hidrodinámicos para evitar pérdidas humanas en este tipo de situaciones.

**II. Terminología**

* **SNDAT:** Sistema Nacional de Detección de Alerta de Tsunamis. Es una entidad encargada de monitorear, predecir y alertar la presencia de tsunamis en las costas colombianas.
* **SATT:** Sistema de Alerta Temprana de Tsunamis. Es el sistema que se propone desarrollar en este proyecto, cuyo objetivo es determinar el nivel de peligro de un evento sísmico con respecto a la generación de un tsunami, establecer zonas de evacuación y asignación de recursos de emergencia/rescate ante la presencia de un tsunami.
* **RSNC:** Red Sismológica Nacional de Colombia. Es la entidad encargada de emitir la información oficial de un evento sísmico.
* **DGR:** Dirección de Gestión de Riesgo. Entidad encargada de difundir información acerca de alertas por tsunamis y desarrollar estrategias de mitigación del riesgo como la evacuación de habitantes de la zona costera afectada.
* **Escenario premodelado:** Modelos matemáticos que permiten con cierta precisión simular y predecir el impacto de los tsunamis en determinados puntos de la costa, tomando como referencia los parámetros sísmicos del evento (altura y velocidad de ola).

**III. Stakeholders**

|  |  |
| --- | --- |
| **Stakeholder** | **Descripción** |
| SNDAT | Sistema Nacional de Detección de Alerta de Tsunamis. Es una entidad encargada de monitorear, predecir y alertar la presencia de tsunamis en las costas colombianas. **El SNDAT es quien encarga la aplicación SATT a este equipo de desarrolladores.** |
| RSNC | Red Sismológica Nacional de Colombia. Es la entidad encargada de emitir la información oficial de un evento sísmico. **Una vez se complete el proyecto, esta entidad enviará dicha información al SATT.** |
| DGR | Dirección de Gestión de Riesgo. Entidad encargada de difundir información acerca de alertas por tsunamis y desarrollar estrategias de mitigación del riesgo como la evacuación de habitantes de la zona costera afectada. **Una vez se complete el proyecto, esta entidad recibirá los boletines de alerta que genere el SATT.** |
| Equipo de Desarrolladores | Corresponde al grupo de ingenieros que desarrollarán el SATT. |

|  |  |
| --- | --- |
| **Stakeholder** | **Expectativas** |
| SNDAT | Se espera un sistema preciso, efectivo y que tenga una adecuada comprensión de los procesos hidrodinámicos para evitar pérdidas humanas causadas por eventos sísmicos que desencadenen tsunamis. Se espera que el sistema sea escalable, de manera que pueda recoger adecuadamente la información de miles de sensores sin perder su disponibilidad. |
| RSNC | Se espera un sistema que mantenga comunicación directa con la RSNC y altamente disponible, de manera que cualquier evento sísmico pueda ser debidamente reportado y se realicen los estudios correspondientes. |
| DGR | Se espera un sistema con un alto desempeño, de manera que el tiempo que transcurre entre que ocurre un evento sísmico, se hacen los estudios correspondientes y se generen boletines de alerta sea mínimo. Esto permitirá mitigar adecuadamente la situación, la cual es labor de esta entidad. |
| Equipo de Desarrolladores | Se espera un sistema altamente modificable y con implementación clara, de manera que se puedan incluir nuevas funcionalidades y pueda ser puesto en manos de otros desarrolladores. Así, se busca que no se dependa exclusivamente de este equipo para realizar modificaciones al software. |

**IV. Análisis de Restricciones**

|  |  |  |
| --- | --- | --- |
| **ID Restricción** | **Tipo** | **Nombre** |
| RES1 | Tecnología | Java |
| **Descripción** | El backend del sistema debe ser desarrollado en el lenguaje Java | |
| **Establecida por** | Equipo de desarrolladores | |
| **Alternativas** | Ninguna | |
| **Observaciones** | Ninguna | |

|  |  |  |
| --- | --- | --- |
| **ID Restricción** | **Tipo** | **Nombre** |
| RES2 | Tecnología | AngularJS – HTML – CSS |
| **Descripción** | El frontend del sistema debe ser desarrollado utilizando el framework AngularJS sobre páginas HTML con páginas de estilos CSS | |
| **Establecida por** | Equipo de desarrolladores | |
| **Alternativas** | Ninguna | |
| **Observaciones** | Ninguna | |

|  |  |  |
| --- | --- | --- |
| **ID Restricción** | **Tipo** | **Nombre** |
| RES3 | Tecnología | Plataforma Cloud – Proveedor BD |
| **Descripción** | El sistema deberá ser desplegado en una plataforma cloud y la información deberá ser almacenada en bases de datos de un proveedor externo, no propias. | |
| **Establecida por** | SNDAT | |
| **Alternativas** | Ninguna | |
| **Observaciones** | Ninguna | |

|  |  |  |
| --- | --- | --- |
| **ID Restricción** | **Tipo** | **Nombre** |
| RES4 | Tecnología | Heroku – MongoDB |
| **Descripción** | Se hará uso de Heroku como servidor de aplicaciones y MongoDB como proveedor de bases de datos NoSQL. | |
| **Establecida por** | Equipo de desarrolladores | |
| **Alternativas** | Ninguna | |
| **Observaciones** | Ninguna | |

|  |  |  |
| --- | --- | --- |
| **ID Restricción** | **Tipo** | **Nombre** |
| RES5 | Tecnología | Geolocalización |
| **Descripción** | El sistema debe poder recibir y procesar datos de localización (latitud y longitud). En particular debe procesar la información de geolocalización enviada por los sensores marítimos y los nuevos eventos para generar alertas en zonas determinadas. | |
| **Establecida por** | SNDAT | |
| **Alternativas** | Ninguna | |
| **Observaciones** | Para la implementación se puede hacer uso de una librería o puede ser desarrollada por el mismo equipo. | |

|  |  |  |
| --- | --- | --- |
| **ID Restricción** | **Tipo** | **Nombre** |
| RES6 | Negocio | Recursos |
| **Descripción** | El sistema deberá ser desarrollado en no más de 4 meses a partir de enero. El equipo contará con 4 integrantes. No se contarán con recursos monetarios destinados al desarrollo de la aplicación. | |
| **Establecida por** | SNDAT | |
| **Alternativas** | Ninguna | |
| **Observaciones** | Ninguna | |

**V. Análisis de Requerimientos Funcionales**

|  |  |
| --- | --- |
| **Identificador** | RF1 |
| **Nombre Requerimiento** | Recibir información de evento sísmico |
| **Actor(es)** | Ninguno |
| **Indispensable/Deseable** | Indispensable |
| **Prioridad** | Alta |
| **Visible/No Visible** | No Visible |
| **Autor** | Mariana Villamizar |
| **Fecha de Elaboración** | 11-02-2016 |
| **Revisado por** |  |
| **Resumen** | El SATT debe recibir información de un evento sísmico registrado por la RSNC. |
| **Entradas** | Localización del evento (latitud y longitud) y distancia del evento a la costa |
| **Pre-Condiciones** | El SATT debe estar inicializado |
| **Post-Condiciones** | Ninguna |
| **Resultados** | Se guardó la información del evento sísmico |
| **Curso básico de eventos** | 1. Se recibe un HTTP Request (POST) de un evento sísmico  2. Se crea un objeto con dicha información  3. Se almacena el objeto en la base de datos |
| **Caminos alternativos** | Ninguno |
| **Caminos de excepción** | La información del evento sísmico no respeta el formato esperado por el SATT |

|  |  |
| --- | --- |
| **Identificador** | RF2 |
| **Nombre Requerimiento** | Recibir información de sensor |
| **Actor(es)** | Ninguno |
| **Indispensable/Deseable** | Indispensable |
| **Prioridad** | Alta |
| **Visible/No Visible** | No Visible |
| **Autor** | Mariana Villamizar |
| **Fecha de Elaboración** | 11-02-2016 |
| **Revisado por** |  |
| **Resumen** | El SATT debe recibir información de cada uno de los 4000 sensores del SNDAT. Cada sensor envía la medición de la altura y velocidad de las olas en el lugar de su ubicación. Esta señal es enviada cada minuto. |
| **Entradas** | Posición (latitud y longitud) y altura y velocidad de ola |
| **Pre-Condiciones** | El SATT debe estar inicializado |
| **Post-Condiciones** | Ninguna |
| **Resultados** | Se guardó la información de cada uno de los sensores, cada minuto. |
| **Curso básico de eventos** | 1. Se recibe un HTTP Request (POST) de un sensor  2. Se crea un objeto con dicha información  3. Se almacena el objeto en la base de datos |
| **Caminos alternativos** | Ninguno |
| **Caminos de excepción** | La información del sensor no respeta el formato esperado por el SATT |

|  |  |
| --- | --- |
| **Identificador** | RF3 |
| **Nombre Requerimiento** | Crear boletín de alerta |
| **Actor(es)** | Ninguno |
| **Indispensable/Deseable** | Indispensable |
| **Prioridad** | Alta |
| **Visible/No Visible** | Visible |
| **Autor** | Mariana Villamizar |
| **Fecha de Elaboración** | 11-02-2016 |
| **Revisado por** |  |
| **Resumen** | El SATT debe crear un boletín de alerta a partir de la información recogida por los sensores y la recibida acerca del evento sísmico |
| **Entradas** | Localización del sensor (latitud y longitud), distancia del sensor a la costa, altura y velocidad de ola (información del sensor y del evento sísmico) |
| **Pre-Condiciones** | El SATT debe estar inicializado |
| **Post-Condiciones** | Se creó una alerta |
| **Resultados** | Se generó un boletín de alerta y se almacenó en la base de datos |
| **Curso básico de eventos** | 1. Se busca el sensor más cercano al evento sísmico  2. Se consulta la información del sensor (posición y altura y velocidad de ola)  3. A partir de la posición del sensor se encuentra la zona costera más cercana  4. A partir de la velocidad de ola y la distancia del evento a la costa se calcula el tiempo de llegada de la ola  5. Teniendo la zona, altura de ola y tiempo de llegada se comparan dichos valores con los de los escenarios premodelados.  6. A partir del escenario más adecuado se crea el boletín de alerta con el perfil de alerta, zona(s) geográfica(s), tiempo de llegada y altura de ola |
| **Caminos alternativos** | Ninguno |
| **Caminos de excepción** | La información del sensor no respeta el formato esperado por el SATT |

|  |  |
| --- | --- |
| **Identificador** | RF4 |
| **Nombre Requerimiento** | Realizar seguimiento a evento sísmico |
| **Actor(es)** | Ninguno |
| **Indispensable/Deseable** | Indispensable |
| **Prioridad** | Alta |
| **Visible/No Visible** | Visible |
| **Autor** | Mariana Villamizar |
| **Fecha de Elaboración** | 11-02-2016 |
| **Revisado por** |  |
| **Resumen** | Una vez generada una alarma, el SATT debe realizar un seguimiento revisando la información de altura de ola registrada por el sensor más cercano al evento sísmico, y debe generar un nuevo boletín de alerta si la diferencia entre la altura de ola que generó la alerta y la altura de ola actual es mayor a 1.5 metros. |
| **Entradas** | Localización del sensor (latitud y longitud), distancia del evento a la costa, altura y velocidad de ola (información del sensor y del evento sísmico), zona |
| **Pre-Condiciones** | El SATT debe estar inicializado |
| **Post-Condiciones** | Se creó una alerta |
| **Resultados** | Se generó un boletín de alerta y se almacenó la alerta en la base de datos |
| **Curso básico de eventos** | 1. Cada 5 minutos se procesa información del sensor más cercano al evento sísmico.  2. Si la diferencia entre la altura de ola que generó la alerta y la altura de ola actual es mayor a 1.5 metros, se continúa al paso 3. De lo contrario, se realiza el paso 1 de nuevo.  3. A partir de la velocidad de ola y la distancia del evento a la costa se calcula el tiempo de llegada de la ola  4. Teniendo la zona, altura de ola y tiempo de llegada se comparan dichos valores con los de los escenarios premodelados.  5. A partir del escenario más adecuado se genera un nuevo boletín de alerta con el perfil de alerta, zona(s) geográfica(s), tiempo de llegada y altura de ola.  6. Si el perfil de alerta del nuevo boletín es informativo, se finaliza el proceso. De lo contrario, se regresa al paso 1. |
| **Caminos alternativos** | Ninguno |
| **Caminos de excepción** | La información del sensor no respeta el formato esperado por el SATT |

**VI. Análisis de Requerimientos No Funcionales**

|  |  |  |  |
| --- | --- | --- | --- |
| **Escenario de calidad #** | 001 | **Stakeholder:** | DGR |
| **Atributo de calidad** | Desempeño | | |
| **Justificación** | Cuando un evento sísmico ocasiona un tsunami, el tiempo es determinante en la mitigación de los daños que este fenómeno pueda causar. Es por esto que los boletines de alerta deben ser generados en el menor tiempo posible. | | |
| **Fuente** | RSNC | | |
| **Estímulo** | Ocurrencia de un evento sísmico | | |
| **Artefacto** | Sistema | | |
| **Ambiente** | Operación normal o estrés, dependiendo de la magnitud del evento sísmico | | |
| **Respuesta** | Se envía un boletín de alerta al DGR | | |
| **Medida de la respuesta** | Entre la recepción de los parámetros sísmicos enviados por la Red Sismológica Nacional de Colombia (RSNC) y el envío del boletín a la Dirección de Gestión del Riesgo (DGR) no deben haber más de 500 milisegundos. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Escenario de calidad #** | 002 | **Stakeholder:** | DGR |
| **Atributo de calidad** | Desempeño | | |
| **Justificación** | Es importante actualizar la información del boletín de alerta, en caso de que la probabilidad de tsunami aumente o disminuya, y de esta forma poder adaptar las acciones de mitigación. | | |
| **Fuente** | DGR | | |
| **Estímulo** | Recepción del boletín de alerta | | |
| **Artefacto** | Sistema | | |
| **Ambiente** | Operación normal o estrés, dependiendo de la magnitud del evento sísmico y la probabilidad de tsunami | | |
| **Respuesta** | Se envía un boletín de alerta actualizado al DGR | | |
| **Medida de la respuesta** | Una vez se ha generado una alarma y si se detecta un cambio en la altura de ola superior a 1.5 m, se debe enviar un nuevo reporte en menos de 500 milisegundos. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Escenario de calidad #** | 003 | **Stakeholder:** | SNDAT |
| **Atributo de calidad** | Escalabilidad | | |
| **Justificación** | Es importante contar con una extensa red de sensores para monitorear el estado de las olas que puedan convertirse en tsunamis, por lo cual se debe poder aumentar la extensión de dicha red sin deteriorar la calidad del sistema. | | |
| **Fuente** | Sensores | | |
| **Estímulo** | Están programados para enviar información cada minuto | | |
| **Artefacto** | Sistema | | |
| **Ambiente** | Operación normal o estrés, dependiendo de si hubo un evento sísmico o tsunami | | |
| **Respuesta** | La información se almacena exitosamente en el historial del sistema | | |
| **Medida de la respuesta** | En una ventana de tiempo de 1 minuto se envía información de todos los sensores ubicados en el borde costero (i.e., 4000), se espera que el sistema trate la información proveniente de cada sensor en menos de 1 minuto. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Escenario de calidad #** | 004 | **Stakeholder:** | SNDAT y DGR |
| **Atributo de calidad** | Disponibilidad | | |
| **Justificación** | La emisión de un boletín de alerta oportunamente es de gran importancia en el manejo de la emergencia y es determinante para la vida de los ciudadanos. Es por esto que el SATT debe poder continuar normalmente su funcionamiento incluso si falla el servidor principal o su base de datos. | | |
| **Fuente** | Ocurrencia de un evento sísmico | | |
| **Estímulo** | Falla en el servidor o base de datos principal | | |
| **Artefacto** | Sistema / Base de datos | | |
| **Ambiente** | Operación normal o estrés, dependiendo de si hubo un evento sísmico o tsunami, o dependiendo del número de solicitudes que esté recibiendo el sistema | | |
| **Respuesta** | El sistema continúa su funcionamiento haciendo uso de los servidores de apoyo o de la base de datos secundaria | | |
| **Medida de la respuesta** | Se espera que el 99.95% de las peticiones hechas al SATT (recepción de información sísmica y olas) sean atendidas | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Escenario de calidad #** | 005 | **Stakeholder:** | SNDAT y DGR |
| **Atributo de calidad** | Seguridad | | |
| **Justificación** | La integridad de los datos es de gran importancia para el sistema porque información falsa puede inducir a boletines de alerta erróneos que a su vez generen una respuesta en cuerpos de rescate y emergencias, todo a raíz de una falsa alarma. | | |
| **Fuente** | Ocurrencia de un evento sísmico | | |
| **Estímulo** | Envío de datos del sensor al SATT o del SATT al browser del usuario | | |
| **Artefacto** | Sistema | | |
| **Ambiente** | Operación normal | | |
| **Respuesta** | Se desecha el dato corrupto del sistema (es decir, no se guarda en la base de datos) y se continúa con el funcionamiento normal de la aplicación | | |
| **Medida de la respuesta** | Debe garantizarse que el 100% de los datos que entran y salen del SATT no son manipulados o modificados. Es decir, no puede darse envío/recepción de información falsa que induzca a boletines de alerta erróneos o información falsa de sensores. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Escenario de calidad #** | 006 | **Stakeholder:** | SNDAT y DGR |
| **Atributo de calidad** | Seguridad | | |
| **Justificación** | La confidencialidad de los datos es de gran importancia para el sistema porque el acceso no autorizado puede afectar aspectos de seguridad nacional, ya que la información que se maneja en el sistema es información sensible sobre posibles emergencias nacionales. | | |
| **Fuente** | Ocurrencia de un evento sísmico | | |
| **Estímulo** | Envío de datos del sensor al SATT o del SATT al browser del usuario | | |
| **Artefacto** | Sistema | | |
| **Ambiente** | Operación normal | | |
| **Respuesta** | Se niega el acceso al usuario no autorizado y se continúa con el funcionamiento normal del sistema. | | |
| **Medida de la respuesta** | No se deben reportar accesos no autorizados al sistema, es decir, deben haber 0 accesos autorizados en cualquier lapso de tiempo. | | |

**VII. Diseño de elementos de Arquitectura**

* **Capa de Presentación**

Se hará uso del framework de Javascript AngularJS 1 para aplicaciones Web de manera que se pueda ofrecer una interfaz usable, con elementos de responsiveness. Esta capa se comunicará con la capa de Negocio por medio del protocolo REST y peticiones HTTP. Adicionalmente, una de sus capacidades es instanciar objetos JSON que pueden ser utilizados y manipulados desde el frontend.

* **Negocio**

Se manejara una arquitectura containerless, lo cual permitirá mayor flexibilidad y rapidez en la etapa de desarrollo de la aplicación. Se utilizará Jersey para instanciar y persistir los DTOs a partir de los objetos JSON recibidos de la capa de presentación y sus atributos. Se utilizará Grizzly para tomar ventaja del API NB de Java, que permite manejar las peticiones asincrónicamente, sin bloquear el servidor por una petición no resuelta. Esto nos permitirá implementar un sistema altamente escalable y con índice reducido latencia.

* + **Capa de Servicios**

Se desarrollará de manera que exponga servicios REST que puedan ser accedidos por la capa de Presentación. Al tratarse de servicios stateless, no se deberán implementar mecanismos para controlar las sesiones de los usuarios, ni almacenar elementos en memoria. Además, esto permitirá aumentar la modificabilidad del sistema en la medida en que REST es un estilo de arquitectura ampliamente utilizado y de implementación sencila, por lo cual la comunicación con otros sistemas no aportará complicaciones. El único *media type* que se manejará será JSON, tanto para entrada como para salida.

* + **Capa de Lógica**

La lógica de la aplicación se desarrollará en el lenguaje Java 8, lo que permitirá desplegar la aplicación en un servidor cloud que es capaz de interpretar este lenguaje. Al tratarse de Java 8, la mejorada eficiencia de los algoritmos permitirá un mejor desempeño de la aplicación. Se implementarán objetos DAO que permitirán realizar operaciones con la base de datos, sin saturar a la capa de servicios con operaciones de persistencia de bajo nivel. Lo anterior permitirá obtener entidades desacopladas que aumentarán la modificabilidad del sistema.

* **Capa de Persistencia**

Se utilizará MongoDB como base de datos no relacional, ya que permite almacenar objetos como celdas de una tabla. De esta manera, por ejemplo, se facilita la búsqueda de sensores y el almacenamiento de su información, permitiendo acceder más fácilmente a un historial.

En la implementación del sistema, la separación por capas se evidenciará en la separación del código por paquetes. Cada paquete será cohesivo, pues las clases que se incluyan en este desempeñarán tareas específicas a la capa a la que pertenecen. La convención de nombramiento tanto de los paquetes como de las clases también reflejará la separación por capas, siendo evidente qué paquete conforma la capa de lógica, qué paquete conforma la capa de servicios y qué paquete conforma la capa de persistencia. Así mismo, la capa de presentación se encontrará en un proyecto distinto y consumirá los servicios prestados por el proyecto de backend.

**VIII. Diseño de arquitectura de solución**

Sistema de Alerta Temprana de Tsunamis



Capa de Presentación

Sistema de eventos sísmicos

Sistema de sensores

MongoDB

REST

REST

REST

Capa de Persistencia

Negocio

Capa de Lógica

Capa de Servicios

**IX. Pruebas de calidad (Experimento 1)**

**Pre-experimentación**

Para verificar el buen funcionamiento del SATT propuesto en el proyecto y el cumplimiento de los requerimientos no funcionales propuestos, se realizaron varias pruebas en las que midieron los tiempos de respuesta al someter el sistema a diferentes cargas de peticiones. La problemática consiste en poner a prueba el funcionamiento del sistema SATT en cuanto al procesamiento y monitoreo de boletines de alerta a partir de eventos sísmicos y el procesamiento de la información de los sensores del sistema. El objetivo de los experimentos a realizar es revisar el comportamiento del sistema frente a los escenarios de calidad 001, 002 y 003 propuestos por los stakeholders y verificar el cumplimiento de dichos escenarios de calidad. Para realizar los experimentos, se realizarán varias iteraciones consumiendo los servicios de la aplicación que tienen que ver con los escenarios de calidad incrementando gradualmente el número de threads y registrando el tiempo de respuesta medio (ms), % de error y rendimiento (threads/sec). A partir de los datos y los resultados obtenidos, se concluirá acerca del cumplimiento de los escenarios de calidad teniendo en cuenta las medidas de respuesta de los mismos. Para llevar a cabo las pruebas, fue necesaria la creación de 4000 sensores que posteriormente se utilizarán para la actualización de los mismos y la verificación del escenario de calidad 003. Ademas, para la creación y actualización de cada uno de ellos se usó un script en el JSon del request para modificar los datos de dicho JSon cada vez que se realiza una petición.

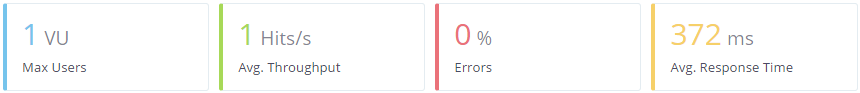
Para realizar los experimentos se utilizó la herramienta JMeter, el cual es un software en el que se pueden hacer pruebas variando la cantidad de threads, el tiempo de ramp-up, y algunos otros parámetros. La característica más importante de esta herramienta es que presenta la posibilidad de utilizar scripts adentro del *request* JSon de manera que se varíen automáticamente algunos de los parámetros del mismo, presentando más posibilidades de experimentación. Adicionalmente, se utilizó la herramienta BlazeMeter integrada como un Add-on a la aplicación desplegada en el cloud usando Heroku. La mayor ventaja de BlazeMeter es la facilidad para modificar los parámetros de la prueba y la variedad de parámetros modificables que presenta. Por ejemplo, se puede agregar un delay a la prueba que consiste en agregar un retraso entre los envios de requests. Esto representa una gran utilidad en cuanto a la base de datos debido a que al enviar una gran cantidad de peticiones en un intervalo de tiempo muy reducido se corre el riesgo de que las peticiones se encolen y se presenten excepciones del tipo Connection Timed Out.

Debido a que la base de datos y el servidor están en la misma ubicación geográfica se espera que las peticiones, ya sea de POST evento sísmico y PUT sensores, presenten poca latencia. Además, gracias al script mencionado anteriormente, se logra actualizar un sensor diferente por thread, logrando que no se encolen peticiones por recursos ocupados.

**Post-experimentación**

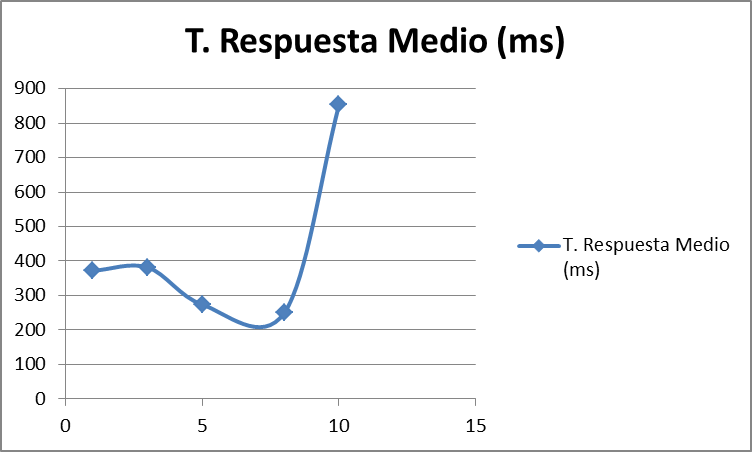
En primera instancia, las pruebas de carga para el escenario de calidad 001 y 002 se realizaron utilizando BlazeMeter debido a que, por medio del cluster de JMeter en las máquinas virtuales, se saturó la base de datos y se reconocieron a las máquinas virtuales como fuentes de posibles ataques. Por lo anterior, las máquinas virtuales *slaves* fueron bloqueadas y quedaron inactivas, sin conexión a internet por un tiempo.

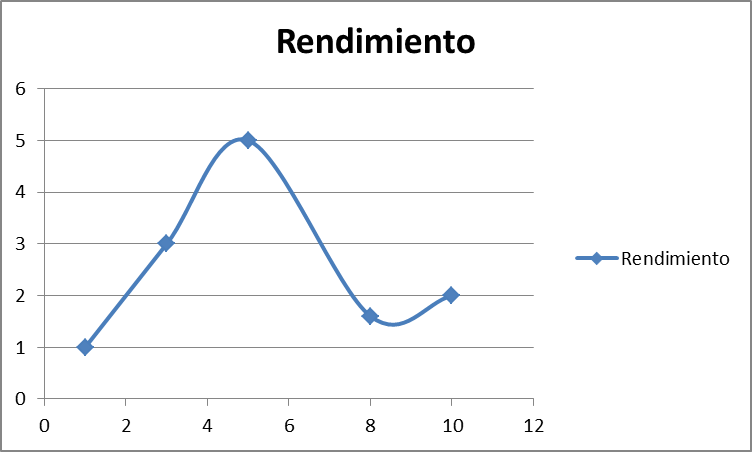
Primero se midió el desempeño de la aplicación en cuanto al manejo de reportes de eventos sísmicos. Para esto, se envió un POST de un evento sísmico y se revisó el tiempo de respuesta del mismo. Este incluye toda la lógica de negocio y la respectiva alerta. Adicionalmente, al hacer la actualización de un evento sísmico se consume el mismo servicio, por lo tanto, la prueba realizada cubre ambos escenarios de calidad (001 y 002).



Como se puede ver, se cumple el requerimiento no funcional ya que al enviar un evento sísmico el sistema responde con una alerta en 372 ms. Además, se presenta un porcentaje de error de 0%, el cual fue verificado a partir de la repetición de la prueba anterior. Posteriormente, se incrementó el número máquinas y los hilos de ejecución por máquina. Esto con el fin de explorar que tantos eventos sísmicos es capaz de soportar el sistema y que tantos errores presenta en respuesta a ellos. Los resultados de las pruebas realizadas son los siguientes:





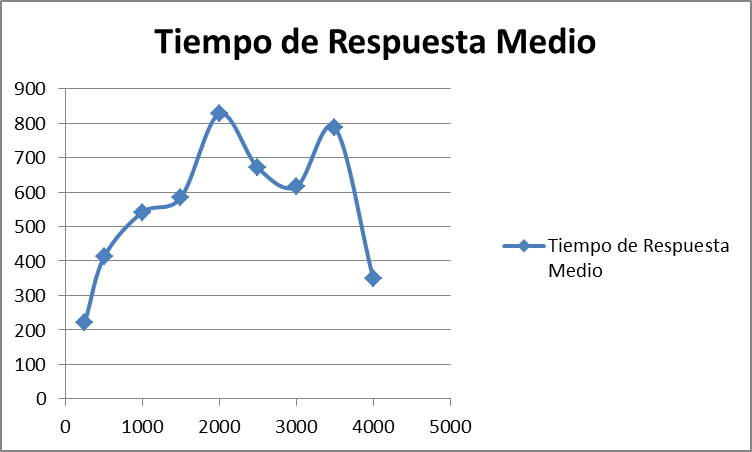


Como se puede ver, el sistema presenta un buen tiempo de respuesta en cuanto a la recepción de un evento sísmico, el análisis del mismo y la publicación de una alerta si es pertinente para una cantidad de threads menor a 8. Sin embargo, al intentar con 10 threads, se presenta un tiempo de respuesta medio de aproximadamente 800 ms. Aunque no se cumple el escenario de calidad, se debe tener en cuenta que la última prueba realizada hace referencia a un caso extremo en el que se presentan 10 eventos sísmicos de gran magnitud en menos de 30 segundos dentro del territorio colombiano. Cabe anotar que a partir de la gráfica de # de threads vs. rendimiento, se puede ver que se obtiene un rendimiento máximo alrededor de los 5 eventos sísmicos. Esto concuerda con la gráfica de # de threads vs. tiempo de respuesta medio en la cual se presenta un tiempo de respuesta mínimo también alrededor de los 5 y 6 eventos sísmicos. Por lo anterior, se concluye que el escenario de calidad 001 se cumple satisfactoriamente dado que en condiciones naturales normales (no apocalípticas) el sistema procesa los eventos sísmicos en menos de 500 ms utilizando toda la lógica de negocio propuesta. Además, también se cumple el escenario de calidad 002 debido a que este consiste en la actualización del boletín de alerta creado a partir de un evento sísmico. Esto se debe a que la actualización del boletín de alerta utiliza la misma lógica de negocio que el análisis inicial de un evento sísmico.

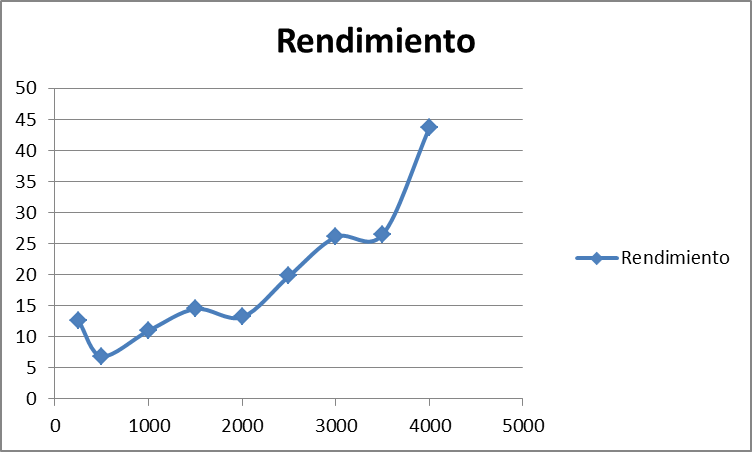
En el caso de los sensores, se hizo una prueba de carga desde el programa JMeter en el que se incrementó el número total de threads gradualmente, donde cada uno hacía una actualización de los sensores del SATT. Para esta prueba se utilizó JMeter principalmente debido a que por medio de scripts incluidos dentro del JSon se pueden modificar los parámetros de este. La prueba consistió en bombardear la aplicación desplegada en Cloud consumiendo el servicio PUT de los sensores y así actualizando una gran cantidad de ellos. Para esto se realizaron las siguientes iteraciones y se consiguieron los siguientes resultados:



Además, en cuanto al tiempo de respuesta medio, se obtuvo la siguiente gráfica:



Adicionalmente, se obtuvo la siguiente gráfica de # de threads vs. Rendimiento



Finalmente, se utilizó el siguiente código JSon para hacer las actualizaciones de los sensores:



Inicialmente, frente a la prueba utilizada en la entrega anterior, hay que identificar un cambio importante: el uso de scripts en el JSon. El script utilizado (${\_\_counter(false, my\_counter)}) utiliza un contador para generar números desde el 1 hasta que se termina la prueba. De esta manera, se generan latitudes y longitudes de 1 a 4000 para actualizar sensores con dichos valores creados anteriormente. Esta modificación fue determinante en el éxito de la prueba debido a que anteriormente se actualizaba siempre el mismo sensor. Esto ocasionaba una excepción de Connection Timmed Out debido a que al llegar demasiadas peticiones para actualizar el mismo sensor la base de datos bloqueaba la tupla y las peticiones se encolaban, al punto que algunas de ellas demoraban más de 30 segundos en ser respondidas y se lanzaba la excepción. A partir de esta modificación hecha, con cada thread se actualiza una entrada diferente de los sensores evitando el problema mencionado anteriormente. Además de lo anterior, fue necesario utilizar cierto tiempo de Ramp-up ya que aunque con la inclusión del script en el JSon se reduce sustancialmente la probabilidad de que se presente la excepción de Connection Timmed Out, si no se utiliza cierto tiempo de Ramp-up existe una pequeña probabilidad de que se encolen algunos de los threads.

En cuanto a los resultados de la prueba, se puede ver que se mantienen unos tiempos de respuesta medios muy reducidos cuando se tienen menos de 1500 threads. Luego hay una franja entre los 2500 y los 3500 threads en la que se incrementa el tiempo de respuesta medio llegando a un máximo de casi 850 ms. Finalmente, cuando se realiza la iteración de mayor peso (la de los 4000 threads que simulan los 4000 sensores necesarios) se puede ver que esta consta de un tiempo de respuesta muy reducido en cuando a las iteraciones exactamente anteriores. Esto podría suceder ya que la primera vez que se accede a un servicio y este consulta la base de datos, el tiempo de respuesta no es muy óptimo comparado con los siguientes accesos. A medida que se incrementó el número de threads de 2000 a 4000 se puede ver que se reduce sustancialmente el tiempo de respuesta medio, posiblemente por lo explicado anteriormente. Además, en la gráfica de # de threads vs. rendimiento se puede que la iteración con los 4000 sensores presenta el rendimiento más elevado con 43.7/sec y esta presenta una tendencia creciente. A partir de lo anterior, y de las tablas y gráficas presentadas, se concluye que el escenario de calidad 003 se cumple debido a que se hace una actualización exitosa de los 4000 sensores en menos de 1 minuto. Aunque en este escenario de calidad no se incluye la métrica *jitter*, es importante notar que los resultados obtenidos al realizar las mediciones variaban incluso con los mismos parámetros. Esto se debe a que las plataformas utilizadas para el despliegue de la aplicación y el almacenamiento de la información son gratis, por lo cual los recursos que se destinan a esta aplicación no se mantienen constantes, y por lo tanto afectan el desempeño de la misma.

**X. Pruebas de calidad (Experimento 2)**

**Pre-experimentación**

Para verificar el buen funcionamiento del SATT propuesto en el proyecto y el cumplimiento de los requerimientos no funcionales propuestos, ahora con la adición del requerimiento no funcional de disponibilidad, se realizaron varias pruebas en las que midieron los tiempos de respuesta al someter el sistema a diferentes cargas de peticiones. La problemática consiste en poner a prueba el funcionamiento del sistema SATT en cuanto al procesamiento y monitoreo de boletines de alerta a partir de eventos sísmicos y el procesamiento de la información de los sensores del sistema. El objetivo de los experimentos a realizar es revisar el comportamiento del sistema frente a los escenarios de calidad 001, 002 y 003 propuestos por los stakeholders y verificar el cumplimiento de dichos escenarios de calidad, implementado diferentes servidores para garantizar la disponibilidad del sistema frente a los escenarios de calidad propuestos. Para realizar los experimentos, se realizarán varias iteraciones consumiendo los servicios de la aplicación que tienen que ver con los escenarios de calidad incrementando gradualmente el número de threads y registrando el tiempo de respuesta medio (ms), % de error y rendimiento (threads/sec). A partir de los datos y los resultados obtenidos, se concluirá acerca del cumplimiento de los escenarios de calidad teniendo en cuenta las medidas de respuesta de los mismos. Para llevar a cabo las pruebas, fue necesaria la creación de 4000 sensores que posteriormente se utilizarán para la actualización de los mismos y la verificación del escenario de calidad 003. Además, para la creación y actualización de cada uno de ellos se usó un script en el JSon del request para modificar los datos de dicho JSon cada vez que se realiza una petición. Adicionalmente, para garantizar la disponibilidad y simular la utilización de cinco servidores que procesan solicitudes en el sistema, fue necesaria la creación de cuatro proyectos más en Heroku en donde se clonó el proyecto original. Esto se hizo debido a que al utilizar la cuenta gratuita de Heroku, este proveedor de cloud provee un solo dyno (servidor). Al tener cinco proyectos estaríamos utilizando cinco dynos, lo que se traduce en cinco servidores. Finalmente, para agregar las funcionalidades de balanceo de carga, se utilizó Nginx. Para incluir los cinco dynos proveídos por Heroku, fue necesario agregar las direcciones IP de cada uno de estos en el archivo de configuraciones de Nginx. De esta manera, se distribuyen las peticiones hechas al sistema a cada uno de los servidores de Heroku. Finalmente, en el archivo de configuraciones de Nginx, se utilizó el método de balanceo de carga Round Robin. A partir de este, se manda la misma cantidad de peticiones a cada servidor sin tener en cuenta que tan saturado este cada uno de ellos. Consideramos que por medio de este método de balanceo de carga se consiguen mejores resultados en cuando a tiempos de respuesta y rendimiento porque no se tiene que consultar el número de conexiones activas a cada servidor cada vez que se haga una petición. Esto se constituye como un beneficio frente a Least Conn debido a que este método si hace verificación del número de conexiones que tiene cada servidor y envía la solicitud al servidor menos saturado.

Para realizar los experimentos se utilizó la herramienta JMeter, el cual es un software en el que se pueden hacer pruebas variando la cantidad de threads, el tiempo de ramp-up, y algunos otros parámetros. La característica más importante de esta herramienta es que presenta la posibilidad de utilizar scripts adentro del request JSon de manera que se varíen automáticamente algunos de los parámetros del mismo, presentando más posibilidades de experimentación.

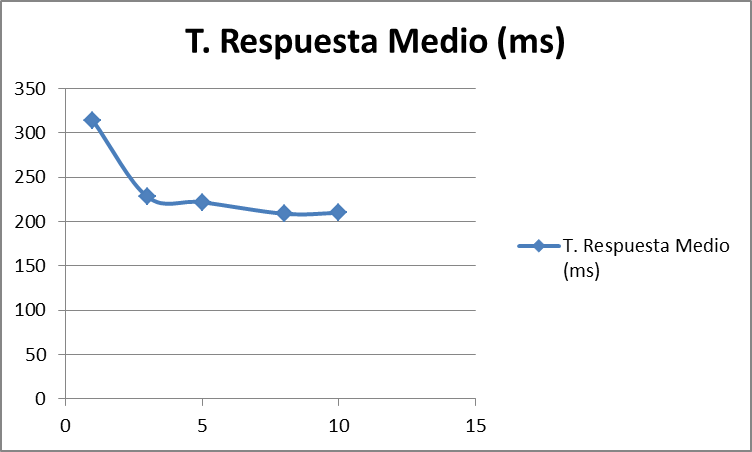
Debido a que la base de datos y el servidor están en la misma ubicación geográfica se espera que las peticiones, ya sea de POST evento sísmico y PUT sensores, presenten poca latencia. Además, gracias al script mencionado anteriormente, se logra actualizar un sensor diferente por thread, logrando que no se encolen peticiones por recursos ocupados.

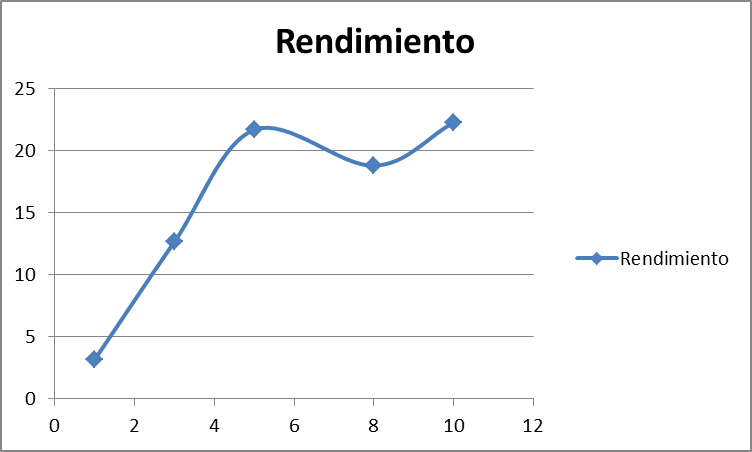
**Post-experimentación**

En primera instancia se realizaron las pruebas de carga para el escenario 001 y 002 utilizando JMetter. Para esta prueba, se utilizó en general un tiempo de ramp up de 0 segundos y se variaron la cantidad de threads por máquina y maquinas totales para de esta manera conseguir diferentes valores de threads totales. Por medio de estas pruebas, primero se midió el desempeño de los escenarios de calidad 001 y 002. Para esto, se envió un POST de un evento sísmico y se revisó el tiempo de respuesta del mismo. Este incluye toda la lógica de negocio y la respectiva alerta. Adicionalmente, al hacer la actualización de un evento sísmico se consume el mismo servicio, por lo tanto, la prueba realizada cubre ambos escenarios de calidad.



Como se puede ver, se cumple el requerimiento no funcional ya que al enviar un evento sísmico el sistema responde con una alerta en 372 ms. Además, se presenta un porcentaje de error de 0%, el cual fue verificado a partir de la repetición de la prueba anterior. Posteriormente, se incrementó el número máquinas y los hilos de ejecución por máquina. Esto con el fin de explorar que tantos eventos sísmicos es capaz de soportar el sistema y que tantos errores presenta en respuesta a ellos.





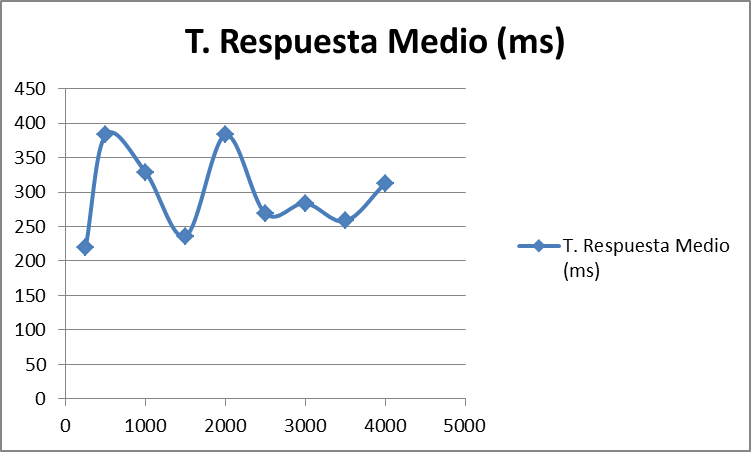
Como se puede ver, en general el sistema presenta un muy buen tiempo de respuesta en cuanto a la recepción de un evento sísmico, el análisis del mismo y la publicación de una alerta si es pertinente. Se presentan buenos tiempos de respuesta hasta para casos extremos en los que se reportan hasta 10 eventos sísmicos en menos de un segundo en zonas costeras cercanas. Para la prueba con un solo thread en el que se prueba el desempeño del sistema se presenta un tiempo de respuesta de 314 ms. En el resto de las pruebas, con tres, cinco, ocho y diez threads, se presentan tiempos de respuesta menores a los 250 ms. Todos estos tiempos resultan satisfactorios debido a que el escenario de calidad presenta una medida de respuesta de máximo 500 ms a partir de la recepción de un evento sísmico, el análisis del mismo y la publicación de una alerta. Por el otro lado, a partir de la gráfica de rendimiento vs # de threads podemos sacar conclusiones importantes acerca del funcionamiento del SATT. Se puede ver que para las primeras pruebas con un menor número de threads se presenta un rendimiento mucho mas reducido que con las pruebas utilizando cinco, ocho y diez threads. Esto guarda concordancia con la gráfica de tiempo de respuesta medio vs # de threads ya que se presentan los mejores tiempos de respuesta con un mayor número de threads. A partir de lo anterior, se puede concluir que se cumple satisfactoriamente el escenario de calidad 001, aun así en condiciones extremas y “apocalípticas”, debido a que se consigue un tiempo de respuesta menor a 500 ms. Adicionalmente, también se cumple el escenario de calidad 002 debido a que este consiste en la actualización del boletín de alerta creado a partir de un evento sísmico, el cual utiliza la misma lógica de negocio que el análisis inicial de un evento sísmico.

En comparación con los resultados de las pruebas realizadas en el experimento 1, se puede ver una gran mejora tanto en tiempo de respuesta y rendimiento. Esto se debe a que la utilización de múltiples servidores y un balanceador de carga permiten el procesamiento de más peticiones concurrentes, lo cual se traduce en mejores tiempos de respuesta y en general un mejor funcionamiento de la aplicación.

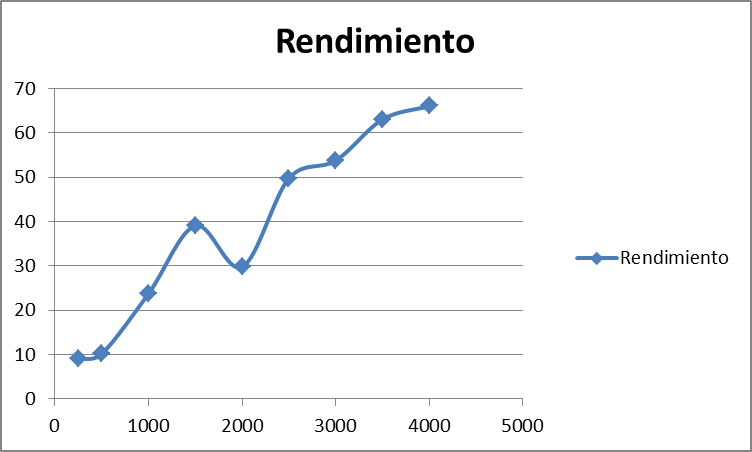
En el caso de los sensores, se hizo una prueba de carga desde el programa JMeter en el que se incrementó el número total de threads gradualmente, donde cada uno hacía una actualización de los sensores del SATT. Para esta prueba se utilizó JMeter principalmente debido a que por medio de scripts incluidos dentro del JSon se pueden modificar los parámetros de este. La prueba consistió en bombardear la aplicación desplegada en Cloud consumiendo el servicio PUT de los sensores y así actualizando una gran cantidad de ellos. Para esto se realizaron las siguientes iteraciones y se consiguieron los siguientes resultados:



Además, en cuanto al tiempo de respuesta medio, se obtuvo la siguiente gráfica:



Adicionalmente, se obtuvo la siguiente gráfica de # de threads vs. Rendimiento:



Finalmente, se utilizó el siguiente código JSon para hacer las actualizaciones de los sensores:



Inicialmente, hay que tener en cuenta el uso de scripts en el JSon de petición. El script utilizado (${\_\_counter(false, my\_counter)}) utiliza un contador para generar números desde el 1 hasta que se termina la prueba. De esta manera, se generan latitudes y longitudes de 1 a 4000 para actualizar sensores con dichos valores creados anteriormente. Esta modificación fue determinante en el éxito de la prueba debido a que al usar valores fijos para la latitud y la longitud el sistema intenta actualizar siempre el mismo sensor. Esto ocasionaba una excepción de Connection Timmed Out debido a que al llegar demasiadas peticiones para actualizar el mismo sensor la base de datos bloqueaba la tupla y las peticiones se encolaban, al punto que algunas de ellas demoraban más de 30 segundos en ser respondidas y se lanzaba la excepción. A partir del uso del script, con cada thread se actualiza una entrada diferente de los sensores evitando el problema mencionado anteriormente. Además de lo anterior, fue necesario utilizar cierto tiempo de Ramp-up ya que aunque con la inclusión del script en el JSon se reduce sustancialmente la probabilidad de que se presente la excepción de Connection Timmed Out, si no se utiliza cierto tiempo de Ramp-up existe una pequeña probabilidad de que se encolen algunos de los threads.

En cuanto a los resultados de la prueba, se puede ver una clara mejora en cuanto a los resultados de las pruebas realizadas en el experimento 1. Se puede ver que en general el tiempo de respuesta medio oscila entre los 200 y 400 ms y que en la prueba con la mayor cantidad de threads (4000) se obitene un tiempo de respuesta medio de 313 ms. Hay que tener en cuenta que en cuanto a las pruebas del experimento 1, en las pruebas realizadas en el experimento 2 en general se redujo el tiempo de ramp. Este cambio fue posible por la inclusión de varias copias de computación y del balanceador de carga debido a que por medio de estos se pueden procesar más solicitudes concurrentemente. Esto se evidencia también en la gráfica de rendimiento vs # de threads. Esta gráfica presenta una tendencia creciente hasta llegar a tener un rendimiento de 66.1 para la prueba con 4000 sensores. Lo anterior, junto con las gráficas, tablas y resultados presentados confirma que el escenario 003 se cumple satisfactoriamente debido a que se hace una actualización exitosa de los 4000 sensores en menos de 1 minuto, posiblemente en menos de 45 segundos.

En conclusión, en cuanto a los demás escenarios de calidad (001, 002 y 003) la inclusión del escenario de calidad 004, que implica una disponibilidad del 99.95% en cuanto a las peticiones hechas al sistema SATT, no afecta los escenarios de calidad mencionados anteriormente. De hecho, a partir de los resultados de las pruebas de este experimento se puede notar una clara mejora en cuando a los tiempos de respuesta y el rendimiento de la recepción, análisis y procesamiento de eventos sísmicos y lecturas de los sensores con respecto a los resultados del experimento 1. Estos resultados nos parecieron sorprendentes debido a que con la inclusión del escenario de calidad 004, que corresponde a la disponibilidad, fue necesario agregar un nodo entre los servidores y las peticiones hechas. Teníamos la hipótesis de que con la adición de este nodo adicional aumentaría la latencia debido a que la comunicación hacia el/los servidor/es no sería directa sino a través de la herramienta Nginx. Sin embargo, este nuevo nodo no representó grandes pérdidas en cuando a tiempos de respuesta, lo cual no afectó la satisfacción de los escenarios de calidad. Por el contrario, con la utilización de Nginx fue posible la adición de más servidores que procesan peticiones concurrentemente y permiten al sistema responder a solicitudes sin que este se sature y sin que se encolen peticiones. Esto es beneficioso para el sistema debido a que se obtiene un menor tiempo de respuesta en cuanto a todas las funcionalidades del sistema y esto, a su vez, permite una reacción más rápida en cuando a terremotos y/o tsunamis, lo cual se traduce en más vidas salvadas en el caso de una catástrofe.

En lo que concierne el escenario de calidad 004 específicamente, en primera instancia es muy poco probable que el sistema como tal presente fallas en cuanto a servidores caídos. Con la implementación de la herramienta Nginx se tienen 5 servidores que tienen la función de back up entre sí de tal manera que si alguno de ellos falla los demás reciben las peticiones que el/los servidor/es caído/s dejó/dejaron de recibir. Adicionalmente, estos 5 servidores están desplegados en cloud usando Heroku. Esto proporciona un mayor grado de confiabilidad en cuando a la disponibilidad del sistema debido a que aun con la versión gratis de Heroku las aplicaciones desplegadas utilizando este servicio en general son muy estables. Cabe anotar que la forma correcta de implementar la disponibilidad en cuanto a servidores consiste en la compra de más dynos a través de Heroku. Sin embargo, esto resulta una solución inviable para el ámbito del curso debido a que es necesario el pago de una cantidad considerable de dinero y no estaría reflejando lo aprendido en el curso.

Por el otro lado, en el caso de la disponibilidad de los datos, para nuestro proyecto se implementó un worker que periódicamente, cada dos horas, lee y copia toda la información de la base de datos principal a una base de datos de respaldo. Además, en la capa del backend de la aplicación, cada vez que se va a hacer una creación, consulta, actualización y eliminación de cualquiera de las entidades de negocio, se verifica la disponibilidad de la base de datos principal. Si esta tiene algún inconveniente en cuanto a la información en ella o al estado de la base de datos, se procesa la solicitud por medio de la base de datos de respaldo. De esta manera, se consigue cierto grado de respuesta a fallos y de integridad en los datos del sistema. De igual manera, en cuanto a la disponibilidad de los datos, Heroku presenta un servicio más efectivo y eficiente para obtener disponibilidad en lo que concierne a las bases de datos. Por medio de la utilización de un ReplicaSet se obtienen backups con intervalos de actualización más reducidos y en general mejores. Sin embargo, al igual que con el caso de los dynos, es necesario el desembolso de una suma considerable de dinero para obtener este servicio a través de Heroku.

**XI. Pruebas de calidad (Experimento 2 Final)**

**Pre-experimentación**

Para verificar el buen funcionamiento del SATT propuesto en el proyecto y el cumplimiento de los requerimientos no funcionales propuestos, ahora con la adición de los requerimientos no funcionales de disponibilidad, seguridad e integridad, se realizaron varias pruebas en las que midieron los tiempos de respuesta al someter el sistema a diferentes cargas de peticiones. La problemática consiste en poner a prueba el funcionamiento del sistema SATT en cuanto al procesamiento y monitoreo de boletines de alerta a partir de eventos sísmicos y el procesamiento de la información de los sensores del sistema. El objetivo de los experimentos a realizar es revisar el comportamiento del sistema frente a los escenarios de calidad propuestos por los stakeholders y verificar el cumplimiento de dichos escenarios de calidad, implementado diferentes servidores para garantizar la disponibilidad del sistema frente a los escenarios de calidad propuestos. Para realizar los experimentos, se realizarán varias iteraciones consumiendo los servicios de la aplicación que tienen que ver con los escenarios de calidad incrementando gradualmente el número de threads y registrando el tiempo de respuesta medio (ms), % de error y rendimiento (threads/sec). A partir de los datos y los resultados obtenidos, se concluirá acerca del cumplimiento de los escenarios de calidad teniendo en cuenta las medidas de respuesta de los mismos. Para llevar a cabo las pruebas, fue necesaria la creación de 4000 sensores que posteriormente se utilizarán para la actualización de los mismos y la verificación del escenario de calidad 003. Además, para la creación y actualización de cada uno de ellos se usó un script en el JSon del request para modificar los datos de dicho JSon cada vez que se realiza una petición. Adicionalmente, para garantizar la disponibilidad y simular la utilización de cinco servidores que procesan solicitudes en el sistema, fue necesaria la creación de cuatro proyectos más en Heroku en donde se clonó el proyecto original. Esto se hizo debido a que al utilizar la cuenta gratuita de Heroku, este proveedor de cloud provee un solo dyno (servidor). Al tener cinco proyectos estaríamos utilizando cinco dynos, lo que se traduce en cinco servidores. Finalmente, para agregar las funcionalidades de balanceo de carga, se utilizó Nginx. Para incluir los cinco dynos proveídos por Heroku, fue necesario agregar las direcciones IP de cada uno de estos en el archivo de configuraciones de Nginx. De esta manera, se distribuyen las peticiones hechas al sistema a cada uno de los servidores de Heroku. Finalmente, en el archivo de configuraciones de Nginx, se utilizó el método de balanceo de carga Round Robin. A partir de este, se manda la misma cantidad de peticiones a cada servidor sin tener en cuenta que tan saturado este cada uno de ellos. Consideramos que por medio de este método de balanceo de carga se consiguen mejores resultados en cuando a tiempos de respuesta y rendimiento porque no se tiene que consultar el número de conexiones activas a cada servidor cada vez que se haga una petición. Esto se constituye como un beneficio frente a Least Conn debido a que este método si hace verificación del número de conexiones que tiene cada servidor y envía la solicitud al servidor menos saturado. Por último, se implementó un artefacto que intercepta todas las llamadas al API del software y verifica, con la información de los headers de cada llamada, el usuario que la está haciendo y el rol del mismo. De esta manera se verifican los permisos que requiere la petición y los compara con los permisos que tiene el usuario que la hace. Ademas, se usa una encriptación aleatoria en base 64 para todos los datos sensibles del usuario como la contraseña y el ID. De esta manera, los datos viajan y se almacenan tanto en el dispositivo como en la base de datos de manera cifrada. En el caso de la integridad de los datos, se implementó un checksum del hash de cada lectura enviada por los sensores de manera que al recibir un dato corrupto este no se tiene en cuenta. De esta manera se busca cumplir los escenarios de calidad 005 y 006.

Para realizar los experimentos se utilizó la herramienta JMeter, el cual es un software en el que se pueden hacer pruebas variando la cantidad de threads, el tiempo de ramp-up, y algunos otros parámetros. La característica más importante de esta herramienta es que presenta la posibilidad de utilizar scripts adentro del request JSon de manera que se varíen automáticamente algunos de los parámetros del mismo, presentando más posibilidades de experimentación.

Debido a que la base de datos y el servidor están en la misma ubicación geográfica se espera que las peticiones, ya sea de POST evento sísmico y PUT sensores, presenten poca latencia. Además, gracias al script mencionado anteriormente, se logra actualizar un sensor diferente por thread, logrando que no se encolen peticiones por recursos ocupados.

**Post-experimentación**

En primera instancia se realizaron las pruebas de carga para el escenario 001 y 002 utilizando JMetter. Para esta prueba, se utilizó en general un tiempo de ramp up de 0 segundos y se variaron la cantidad de threads por máquina y maquinas totales para de esta manera conseguir diferentes valores de threads totales. Por medio de estas pruebas, primero se midió el desempeño de los escenarios de calidad 001 y 002. Para esto, se envió un POST de un evento sísmico y se revisó el tiempo de respuesta del mismo. Este incluye toda la lógica de negocio y la respectiva alerta. Adicionalmente, al hacer la actualización de un evento sísmico se consume el mismo servicio, por lo tanto, la prueba realizada cubre ambos escenarios de calidad.



Como se puede ver, aun con la inclusión de los escenarios de calidad 004, 005 y 006 se cumple el requerimiento no funcional ya que al enviar un evento sísmico el sistema responde con una alerta en 372 ms. Además, se presenta un porcentaje de error de 0%, el cual fue verificado a partir de la repetición de la prueba anterior. Posteriormente, se incrementó el número máquinas y los hilos de ejecución por máquina. Esto con el fin de explorar que tantos eventos sísmicos es capaz de soportar el sistema y que tantos errores presenta en respuesta a ellos.

Como se puede ver, en general el sistema presenta un muy buen tiempo de respuesta en cuanto a la recepción de un evento sísmico, el análisis del mismo y la publicación de una alerta si es pertinente. Se presentan buenos tiempos de respuesta hasta para casos extremos en los que se reportan hasta 10 eventos sísmicos en menos de un segundo en zonas costeras cercanas. Para la prueba con un solo thread en el que se prueba el desempeño del sistema se presenta un tiempo de respuesta de 314 ms. En el resto de las pruebas, con tres, cinco, ocho y diez threads, se presentan tiempos de respuesta menores a los 250 ms. Todos estos tiempos resultan satisfactorios debido a que el escenario de calidad presenta una medida de respuesta de máximo 500 ms a partir de la recepción de un evento sísmico, el análisis del mismo y la publicación de una alerta. Por el otro lado, a partir de la gráfica de rendimiento vs # de threads podemos sacar conclusiones importantes acerca del funcionamiento del SATT. Se puede ver que para las primeras pruebas con un menor número de threads se presenta un rendimiento mucho mas reducido que con las pruebas utilizando cinco, ocho y diez threads. Esto guarda concordancia con la gráfica de tiempo de respuesta medio vs # de threads ya que se presentan los mejores tiempos de respuesta con un mayor número de threads. A partir de lo anterior, se puede concluir que se cumple satisfactoriamente el escenario de calidad 001, aun así en condiciones extremas y “apocalípticas”, debido a que se consigue un tiempo de respuesta menor a 500 ms. Adicionalmente, también se cumple el escenario de calidad 002 debido a que este consiste en la actualización del boletín de alerta creado a partir de un evento sísmico, el cual utiliza la misma lógica de negocio que el análisis inicial de un evento sísmico.

En comparación con los resultados de las pruebas anteriores, se puede ver que el tiempo de respuesta medio aumentó sistemáticamente y de igual manera, el rendimiento disminuyó casi en igual proporción en cada una de las iteraciones de las pruebas. Esto se puede explicar por medio de la implementación de los escenarios de calidad adicionales propuestos para la entrega final del experimento dos. Para cumplir con el requerimiento no funcional de seguridad, es necesario realizar más solicitudes a la base de datos y procesar información a partir de la respuesta de la misma. Esto aumenta el tiempo de respuesta de las peticiones hechas al sistema lo cual de igual manera disminuye el rendimiento del mismo.

En el caso de los sensores, se hizo una prueba de carga desde el programa JMeter en el que se incrementó el número total de threads gradualmente, donde cada uno hacía una actualización de los sensores del SATT. Para esta prueba se utilizó JMeter principalmente debido a que por medio de scripts incluidos dentro del JSon se pueden modificar los parámetros de este. La prueba consistió en bombardear la aplicación desplegada en Cloud consumiendo el servicio PUT de los sensores y así actualizando una gran cantidad de ellos. Para esto se realizaron las siguientes iteraciones y se consiguieron los siguientes resultados:



Además, en cuanto al tiempo de respuesta medio, se obtuvo la siguiente gráfica:

Adicionalmente, se obtuvo la siguiente gráfica de # de threads vs. Rendimiento:

Finalmente, se utilizó el siguiente código JSon para hacer las actualizaciones de los sensores:



Inicialmente, hay que tener en cuenta el uso de scripts en el JSon de petición. El script utilizado (${\_\_counter(false, my\_counter)}) utiliza un contador para generar números desde el 1 hasta que se termina la prueba. De esta manera, se generan latitudes y longitudes de 1 a 4000 para actualizar sensores con dichos valores creados anteriormente. Esta modificación fue determinante en el éxito de la prueba debido a que al usar valores fijos para la latitud y la longitud el sistema intenta actualizar siempre el mismo sensor. Esto ocasionaba una excepción de Connection Timmed Out debido a que al llegar demasiadas peticiones para actualizar el mismo sensor la base de datos bloqueaba la tupla y las peticiones se encolaban, al punto que algunas de ellas demoraban más de 30 segundos en ser respondidas y se lanzaba la excepción. A partir del uso del script, con cada thread se actualiza una entrada diferente de los sensores evitando el problema mencionado anteriormente. Además de lo anterior, fue necesario utilizar cierto tiempo de Ramp-up ya que aunque con la inclusión del script en el JSon se reduce sustancialmente la probabilidad de que se presente la excepción de Connection Timmed Out, si no se utiliza cierto tiempo de Ramp-up existe una pequeña probabilidad de que se encolen algunos de los threads.

En cuanto a los resultados de la prueba, se puede ver que, al igual que en el caso del POST de un evento sísmico, se presentaron aumentos en los tiempos de respuesta medios y disminuciones en los rendimientos de todas las iteraciones realizadas. Aunque se sigue cumpliendo el escenario de calidad, en el que se debe recibir la información de las lecturas de los 4000 sensores en menos de un minuto, se puede ver que la inclusión de los requerimientos de seguridad e integridad afecta el funcionamiento del sistema. Al incluir estos escenarios de calidad, es necesario hacer verificaciones de credenciales y de permisos del usuario y esto se traduce un acceso adicional a la base de datos. Este acceso implica aumentos en los tiempos de respuesta medios de todas las iteraciones de la prueba y, a su vez, disminuciones en el rendimiento.

En conclusión, en cuanto a los escenarios de calidad iniciales (001, 002, 003 y 004) la inclusión de los requerimientos no funcionales 005 y 006, no afecta en gran medida los escenarios de calidad mencionados anteriormente. Aunque en general los tiempos de respuesta medios aumentaron y que los rendimientos de los threads disminuyeron, las métricas de los escenarios de calidad se siguen cumpliendo. El anterior es un trade-off que vale la pena realizar debido a que se sacrificando 0.1 segundos se obtiene un software mucho más confiable en cuanto a la seguridad e integridad de los datos.