**EXPERIMENTO 1 - REPORTE DE RESULTADOS**

**Objetivo**

Medir la escalabilidad y desempeño de una aplicación desarrollada usando un estilo arquitectural asincrónico, asegurando el cumplimiento de unos escenarios de calidad establecidos.

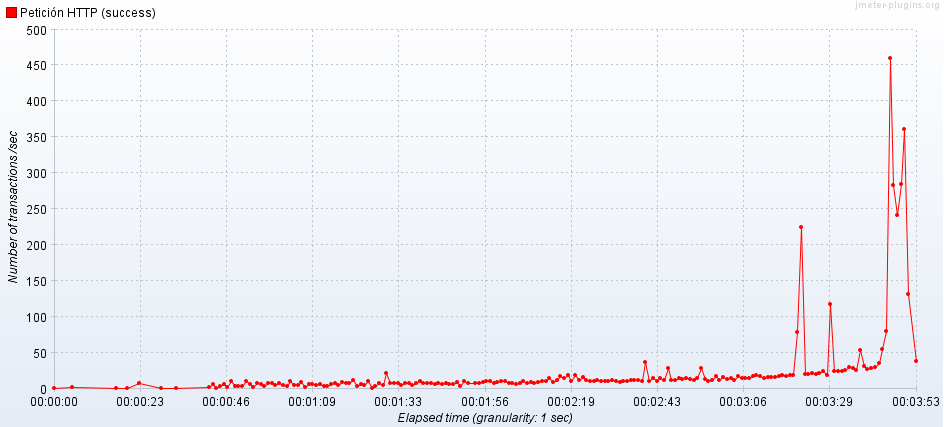
**Escenarios de calidad**

* **Latencia:** Se espera que el tiempo de respuesta de las funcionalidades de la aplicaciones no exceda 1 s.
* **Escalabilidad:** Se espera que el sistema soporte hasta 4500 vehículos (4000 VCubs, 250 tranvías, 250 Mobibus) intentando enviar información (posición, alarma, etc.) al servidor principal en 5 s.

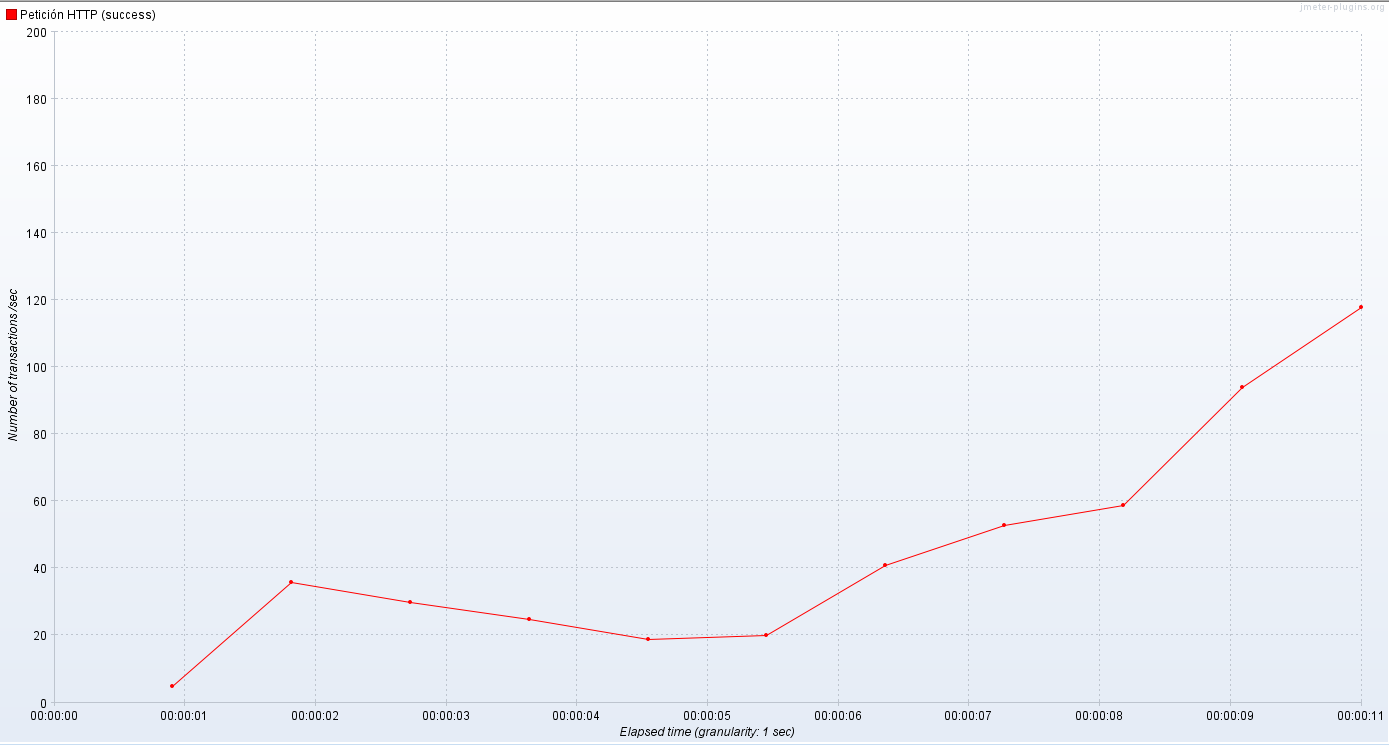
**Resultados experimentación**

**Entrega 1**

* **Pruebas de escalabilidad**
  + Actualización de posición de 4500 vehículos



* + Reporte de accidente de 500 vehículos (mobibuses y tranvías)



* **Pruebas de desempeño**
  + Consultar ubicaciones de vehículos



* + Consultar disponibilidad de vcbus en las estaciones



* + Hacer reserva de movibus



* + Consultar tiempo trayectos

pedirvcub.PNG

* + Restituir vcub

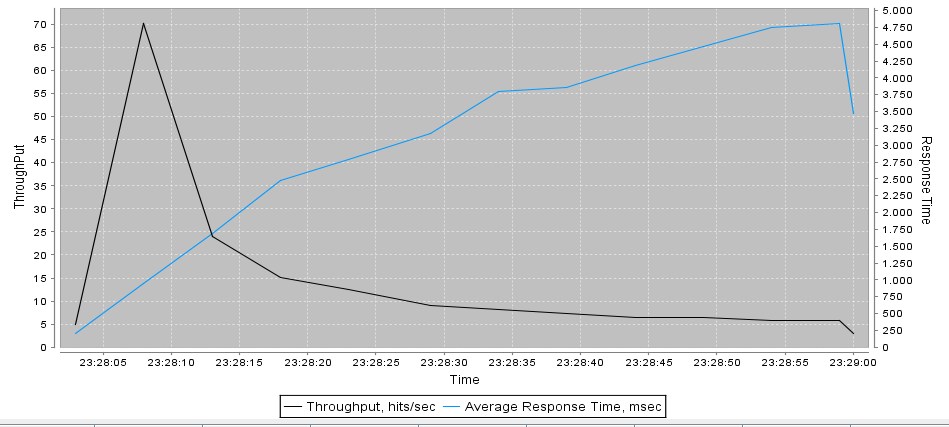
restituir.PNG

**Entrega 2**

* **Pruebas de escalabilidad**

Para probar la escalabilidad del software se sometió a una prueba de estrés, en donde se enviaron 5000 solicitudes casi de inmediato por medio de JMeter y se midió el tiempo de respuesta, el rendimiento y el porcentaje de fallas.

La gran diferencia de esta entrega con respecto a la anterior es que ahora la aplicación corre sobre un servidor en la nube de Heroku. Esto aunque tenga beneficios también implica una restricción pues no se cuenta con recursos ilimitados o completos debido a que se usa una versión de prueba la cual en un punto limita el número de transacciones por segundo, cuando esto ocurre la aplicación empieza a fallas ya responder con mensajes de error.

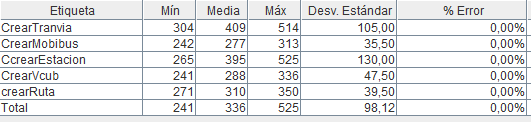


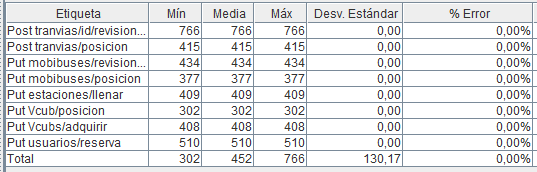
En este caso se vio cómo a partir de un punto hubo un comportamiento extremo. evidenciado en el pico que presenta la gráfica. El tiempo de respuesta fue incrementando a medida que Jmeter enviaba más solicitudes. Otra posible consecuencia diferente a las razones físicas que pueden limitar el desempeño por parte de Heroku, es que el tamaño del json con las posición actualizada que viaja por la red se hace cada vez más pesado a medida que recibe solicitudes. Es entonces una posible justificación de los elevados tiempos de respuesta en las últimas solicitudes, el hecho de que está viajando por la red una cantidad mucho mayor de información. Por otra parte, aunque el tiempo de respuesta fue superior, cabe recalcar que hubo 0% de errores en la prueba de estrés.

* **Pruebas de desempeño**

Para experimentar sobre el desempeño, se realizó una serie de pruebas para cada uno de los servicios que ofrece la aplicación. Estas se hicieron con la base de datos vacía y se ejecutaron primero los servicios de crear, luego los servicios de actualización y, posteriormente los consultas o servicios de get.

Dichas pruebas fueron realizadas con jmeter y a continuación se presentan los resultados obtenidos.







El máximo tiempo de latencia que podemos analizar de los datos fue de 985 para el servicio de get tranvías. por lo que la aplicación cumple con el requerimiento de rendimiento, tiempo de respuesta menor a 1 segundo. También es importante resaltar que el máximo tiempo medio de los servicios es de 766, también del servicio de get tranvías. Lo anterior se asume que es por que las entidades de tranvías se hacen muy pesadas tras haber actualizado varias veces su posición y sus revisiones. Esto es un posible punto a mejorar para el siguiente experimento con el fin de garantizar estabilidad en la aplicación.

Por otra parte, es relevante mencionar todos los tiempos aumentaron, la principal razón es el despliegue de la aplicación en el cloud de Heroku. El aumento en la latencia de debe al tiempo en que la información tiene que viajar desde Colombia hasta las instalaciones de Amazon web services en el este de Estados Unidos, donde actualmente está alojado el backend del grupo, este tiempo se estima segun cloudPing.info en 185ms.