**Documentación de las pruebas Experimento 1, Entrega 1**

**Objetivos**

* Identificar que la cantidad de solicitudes de sensores (transacciones) concurrentes por segundo que puede soportar la aplicación antes de presentar errores o pasarse del tiempo límite es mínimo de 4800 solicitudes.
* Identificar la cantidad de peticiones (transacciones) concurrentes por segundo que puede soportar la aplicación antes de presentar errores en general.
* identificar que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas en el caso.

**CONSTRUCCIÓN DE LA PRUEBA 1 (Recibir 4800 solicitudes de sensores)**

Se construyó una prueba de carga enfocada en el porcentaje de error y el tiempo de respuesta del servicio getSensorTemperatura, ya que se supone que esta es una de las acciones que se ejecutarán con mayor frecuencia (mayor inclusive a la de agregar un sensor). Variamos el número de threads de 1000 a 8000(donde aún no se presentó un porcentaje de error mayor al 0%, ni un tiempo de respuesta superior a 1 segundo), buscando comprobar que se cumpliera la condición de poder atender 4800 solicitudes de sensores simultaneas en una ventana de tiempo de 1 segundo. Se definió un ramp up period de 60 segundos (todas las peticiones debían hacerse en un tiempo uniforme inferior a 60 segundos) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: getSensorTemperatura

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /sensorTemperatura/get

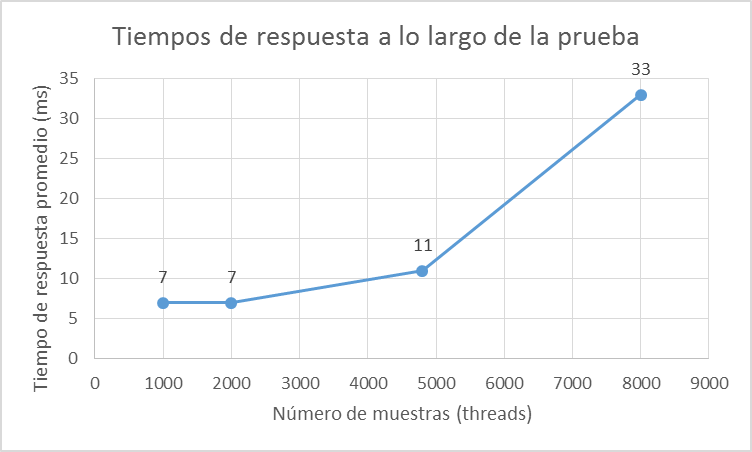
Parameters: vacío pues es un get.

Todo lo anterior cumplió la misión de identificar un rango de valores posibles de saturación de la aplicación construida, y de asegurar que se pueden atender 4800 solicitudes de sensores simultáneas en una ventana de tiempo de 1 segundo.

**RESULTADOS DE LA PRUEBA**

Se identificó que la cantidad de solicitudes de sensores (transacciones) concurrentes por segundo que puede soportar la aplicación antes de presentar errores o pasarse del tiempo límite es superior a 8000 solicitudes concurrentes por segundo, abriendo la posibilidad de la implementación del sistema para los 4800 sensores.

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba**



**Tabla de resumen**



**CONSTRUCCIÓN DE LA PRUEBA 2 (Desempeño y escalabilidad)**

Antes de iniciar el planteamiento de la prueba, identificamos que los escenarios de calidad a verificar son escalabilidad, medido aumentando la cantidad de usuarios concurrentes; y desempeño, compuesto por latencia y rendimiento. Ambos escenarios se ven definidos por el porcentaje de error a su vez.

Así pues, se construyó una prueba de carga enfocada en el porcentaje de error y el tiempo de respuesta de un servicio en general, en este caso getCaudales y postCaudal. Variamos el número de threads de 1000 a 20000 en el caso del post, y a 15000 en el caso del get (donde se presentaron porcentajes de error superiores al 0%) buscando identificar un rango de valores posibles de saturación de la aplicación construida. Se definió un ramp up period de 60 segundos (todas las peticiones debían hacerse en un tiempo uniforme inferior a 60 segundos) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: postCaudal

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /caudal/add

Parameters (JSON):

{ "caudal": 10 }

* Nombre del servicio: getCaudales

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /caudal/get

Parameters: vacío pues es un get

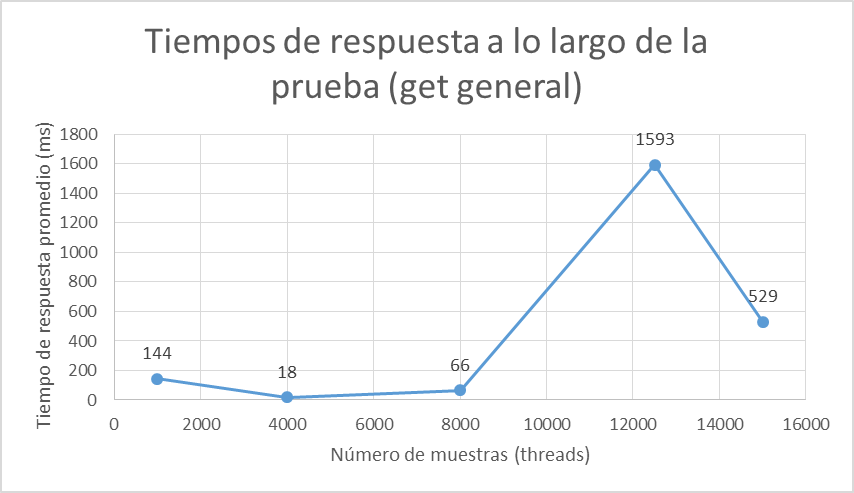
Todo lo anterior cumplió la misión de identificar un rango de valores posibles de saturación de la aplicación construida de un servicio en general en cuanto al get y al post.

**RESULTADOS DE LA PRUEBA**

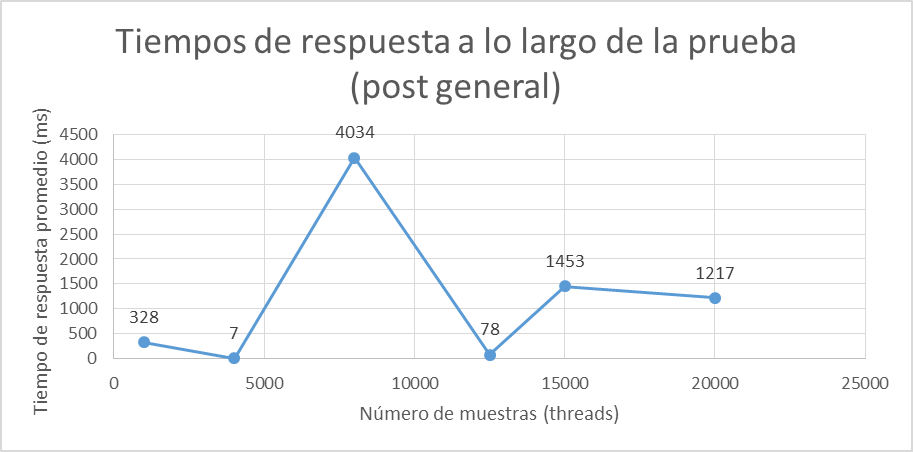
Se identificó que la cantidad de peticiones (transacciones) concurrentes por segundo que puede soportar la aplicación antes de presentar errores es superior a 15000 e inferior a 20000 peticiones por segundo.

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba**

* **Get**

****

* **Post**

****

**Tablas de resumen**

* **Get**



* **Post**



**CONSTRUCCIÓN DE LA PRUEBA 2 parte 1 (Cumplimiento de los requerimientos)**

Se construyó una prueba de carga enfocada en el tiempo de respuesta de los servicios post disponibles ya que se supone que estas acciones presentan una utilización de recursos superior a la de una acción GET. Variamos el número de threads de 90 a 1200 dependiendo de la cantidad máxima de solicitudes que pueden realizar los servicios, un ramp up period de 60 segundos (todas las peticiones debían hacerse en un tiempo uniforme inferior a los 60 segundos) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: postCampo

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /campos/add

Parameters (JSON):

{ "departamento": "Cundinamarca",

"latitud": 124,

"longitud": 125 }

* Nombre del servicio: postCaudal

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /caudal/add

Parameters (JSON):

{ "caudal": 10 }

* Nombre del servicio: postConsumoEnergia

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /consumoEnergia/add

Parameters (JSON):

{ "consumoEnergia": 10 }

* Nombre del servicio: postJefe

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /jefes/addJC

Parameters (JSON):

{ "nombre": "Laura",

"cedula": "420420",

"direccion":"42",

"edad": 4,

"nacionalidad": "Colombiana",

"telefono":"sapo" }

* Nombre del servicio: postPozo

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /pozos/add

Parameters (JSON):

{ "estado": "cerrado",

"latitud": 124,

"longitud": 125 }

* Nombre del servicio: postReporte

Método: POST (porque se consume un servicio POST)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /reportes/add

Parameters (JSON):

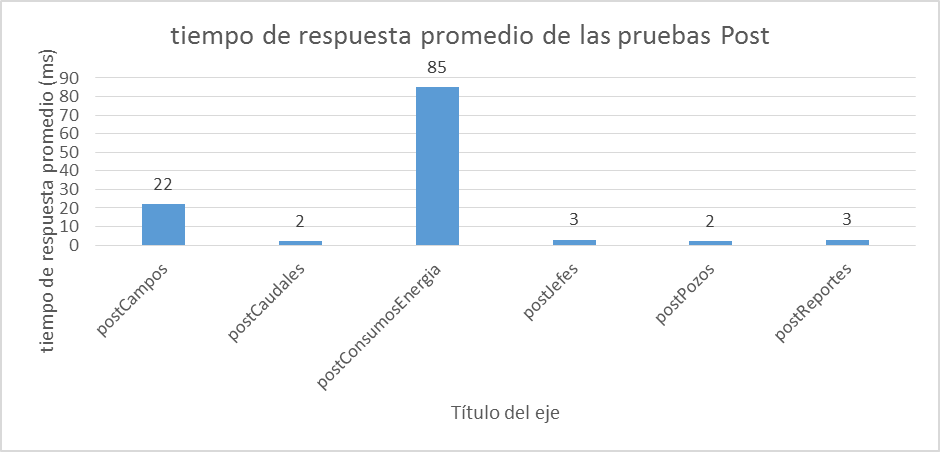
{ "descripcion": "hola"}

Todo lo anterior cumplió la misión de identificar que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas.

**RESULTADOS DE LA PRUEBA**

Se identificó que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas con la solución actual.

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba**

****

**Tabla de resumen**



**CONSTRUCCIÓN DE LA PRUEBA 2 parte 2 (Cumplimiento de los requerimientos)**

Se construyó una prueba de carga enfocada en el tiempo de respuesta de los servicios get disponibles ya que se supone que estas acciones van a ser utilizadas al analizar la información de los censores de forma recurrente, con mayor periodicidad a los servicios post. Variamos el número de threads de 90 a 1200 dependiendo de la cantidad máxima de solicitudes que pueden realizar los servicios un ramp up period de 60 segundos (todas las peticiones debían hacerse en un tiempo uniforme inferior a los 60 segundos) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: getCampos

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /campos/get

Parameters: vacío pues es un get

* Nombre del servicio: getCaudales

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /caudal/get

Parameters: vacío pues es un get

* Nombre del servicio: getConsumosEnergia

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /consumoEnergia/ get

Parameters: vacío pues es un get

* Nombre del servicio: getJefes

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /jefes/getjefecampo

Parameters: vacío pues es un get

* Nombre del servicio: getPozos

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /pozos/get

Parameters: vacío pues es un get

* Nombre del servicio: getReportes

Método: GET (porque se consume un servicio GET)

Nombre del servidor: localhost.

Puerto: 8080.

Ruta: /reportes/get

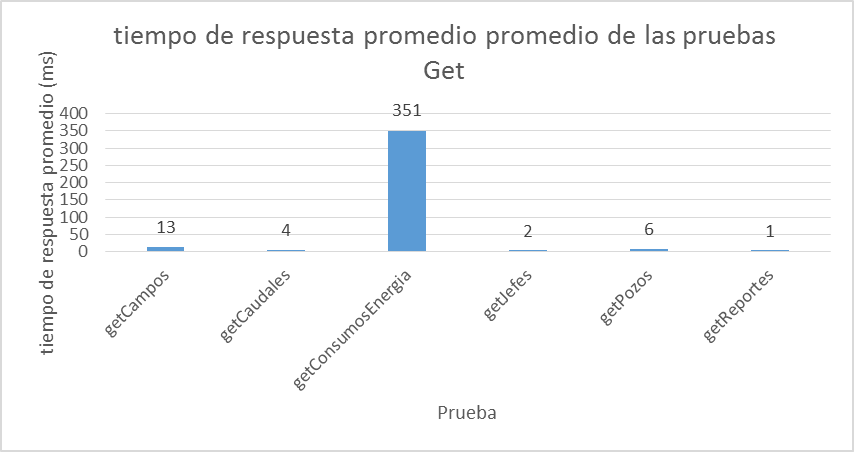
Parameters: vacío pues es un get

Todo lo anterior cumplió la misión de identificar que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas.

**RESULTADOS DE LA PRUEBA**

Se identificó que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas con la solución actual.

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba**

****

**Tabla de resumen**



**Conclusiones:**

* Se halló un rango de valores posibles de saturación de la aplicación construida, y se aseguró que se pueden atender 4800 solicitudes de sensores simultáneas en una ventana de tiempo de 1 segundo.
* Se identificó que la cantidad de peticiones (transacciones) concurrentes por segundo que puede soportar la aplicación antes de presentar errores es superior a 15000 e inferior a 20000 peticiones por segundo.
* Se identificó que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas con la solución actual