**Documentación de las pruebas Experimento 2, 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 que se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas en el caso.
* Comparar los resultados de las entregas experimento 1-2 (entrega 2) y experimento 2-1 (entrega 3) y analizar sus diferencias.

**CONSTRUCCIÓN DE LA PRUEBA 1 parte 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 getReportes, ya que se supone que esta es una de las acciones críticas que podrán ejecutarse al obtener la información de los 4800 sensores. Buscamos comprobar que se cumpliera la condición de poder atender 4800 solicitudes de sensores simultaneas en una ventana de tiempo de 1 segundo ( con dicha cantidad no se presentó un porcentaje de error mayor al 0%, ni un tiempo de respuesta superior a 1 segundo). Se definió un ramp up period de 1 segundo (todas las peticiones debían hacerse en un tiempo uniforme inferior a 1 segundo) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: getReportes

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /reportes

Parameters: vacío pues es un get.

Todo lo anterior cumplió la misión 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 mínimo de 4800 solicitudes concurrentes por segundo.

**Tabla de resumen**



**CONSTRUCCIÓN DE LA PRUEBA 1 parte 2(Enviar instrucciones a los 4800 sensores)**

Se construyó una prueba de carga enfocada en el porcentaje de error y el tiempo de respuesta del servicio postReportes, ya que se supone que esta es una de las acciones críticas que podrán ejecutarse al enviarle instrucciones a los 4800 sensores. Buscamos comprobar que se cumpliera la condición de poder enviar solicitudes a los 4800 sensores de forma simultánea en una ventana de tiempo de 1 segundo (con dicha cantidad no se presentó un porcentaje de error mayor al 0%, ni un tiempo de respuesta superior a 1 segundo). Se definió un ramp up period de 1 segundo (todas las peticiones debían hacerse en un tiempo uniforme inferior a 1 segundo) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: postReporte

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /reportes

Parameters (JSON):

{ "descripcion": "hola"}

Todo lo anterior cumplió la misión 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 que pueden enviarse a los sensores (transacciones) concurrentes por segundo antes de presentar errores o pasarse del tiempo límite es mínimo de 4800 solicitudes concurrentes por segundo.

**Tabla de resumen**



**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, y en un caso a 4800, dependiendo de la cantidad máxima de solicitudes que pueden realizar los servicios, un ramp up period de 1 segundo (todas las peticiones debían hacerse en un tiempo uniforme inferior 1 segundo1) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: postCampo

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /campos

Parameters (JSON):

{ "departamento": "Cundinamarca",

"latitud": 124,

"longitud": 125 }

* Nombre del servicio: postCaudal

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /caudales

Parameters (JSON):

{ "caudal": 10 }

* Nombre del servicio: postConsumoEnergia

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /consumoEnergia

Parameters (JSON):

{ "consumoEnergia": 10 }

* Nombre del servicio: postJefesDeCampo

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /jefesDeCampo

Parameters (JSON):

{ "nombre": "Laura",

"cedula": "420420",

"direccion":"42",

"edad": 4,

"nacionalidad": "Colombiana",

"telefono":"sapo" }

* Nombre del servicio: postJefesAsignado

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /jefesAsignado

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: 172.24.42.64

Puerto: 80

Ruta: /pozos

Parameters (JSON):

{ "estado": "cerrado",

"latitud": 124,

"longitud": 125 }

* Nombre del servicio: postReporte

Método: POST (porque se consume un servicio POST)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /reportes

Parameters (JSON):

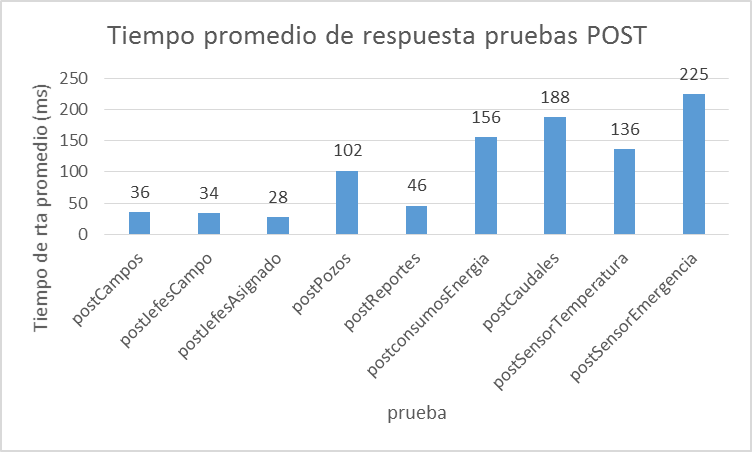
{ "descripcion": "hola"}

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 forma recurrente, con mayor periodicidad a los servicios post. Variamos el número de threads de 90 a 1200, y hasta 4800 en un caso, dependiendo de la cantidad máxima de solicitudes que pueden realizar los servicios un ramp up period de 1 segundo (todas las peticiones debían hacerse en un tiempo uniforme inferior a 1 segundo) lanzando cada grupo de hilos separadamente y sin iteraciones.

* Nombre del servicio: getCampos

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /campos

Parameters: vacío pues es un get

* Nombre del servicio: getCaudales

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /caudales

Parameters: vacío pues es un get

* Nombre del servicio: getConsumosEnergia

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /consumoEnergia

Parameters: vacío pues es un get

* Nombre del servicio: getJefesDeCampo

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /jefesDeCampo

Parameters: vacío pues es un get

* Nombre del servicio: getJefesAsignado

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /jefesAsignado

Parameters: vacío pues es un get

* Nombre del servicio: getPozos

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /pozos

Parameters: vacío pues es un get

* Nombre del servicio: getReportes

Método: GET (porque se consume un servicio GET)

Nombre del servidor: 172.24.42.64

Puerto: 80

Ruta: /reportes

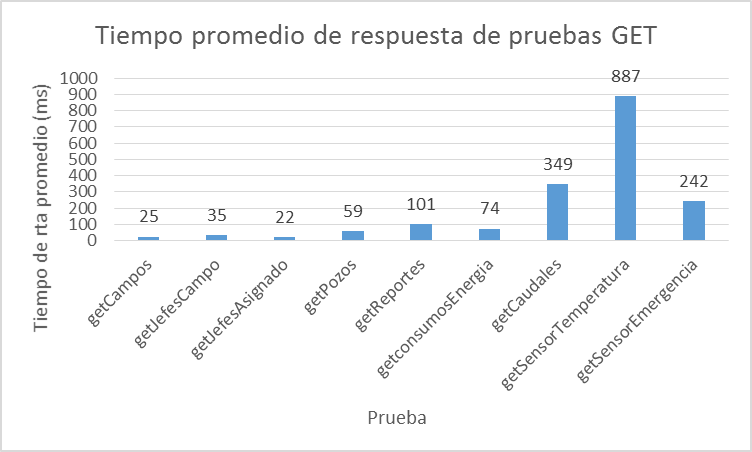
Parameters: vacío pues es un get

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**



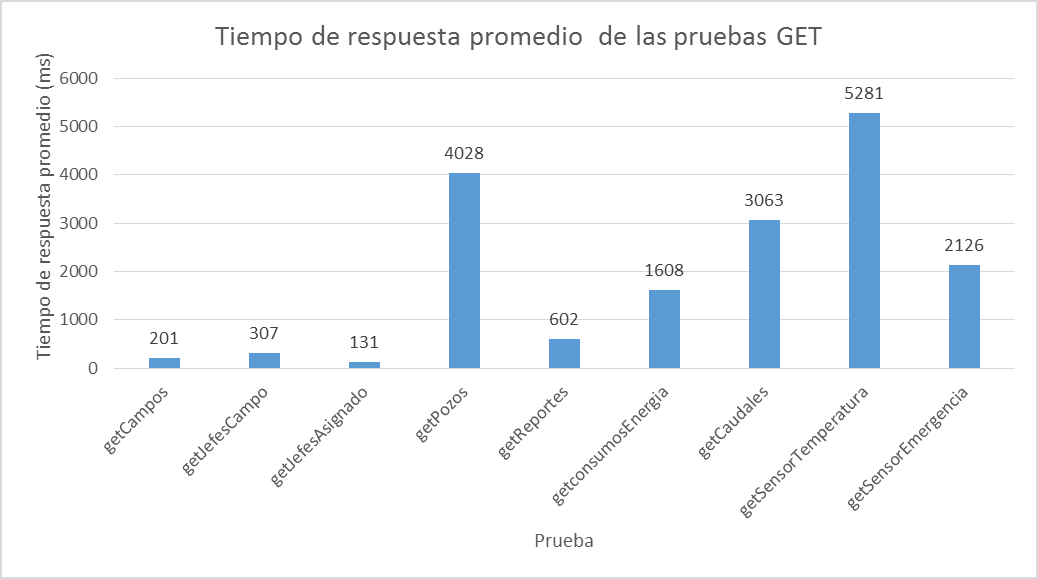
**COMPARACIÓN DE LOS RESULTADOS ENTRE ENTREGAS**

Para esta entrega se implementaron herramientas para ayudar al atributo de calidad de disponibilidad.

A continuación se observan las tablas de resumen de la entrega 2 para los servicios GET y POST, junto a sus gráficos de tiempo de respuesta promedio.

**GET**

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba 2**

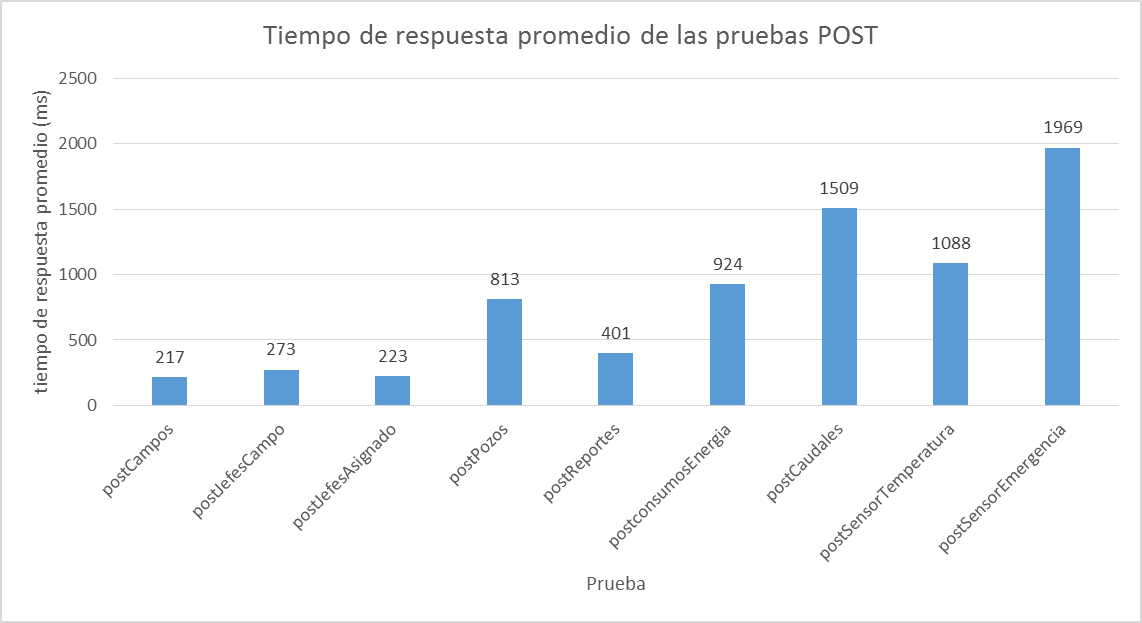
****

**Tabla de resumen2**



**POST**

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba 2**

****

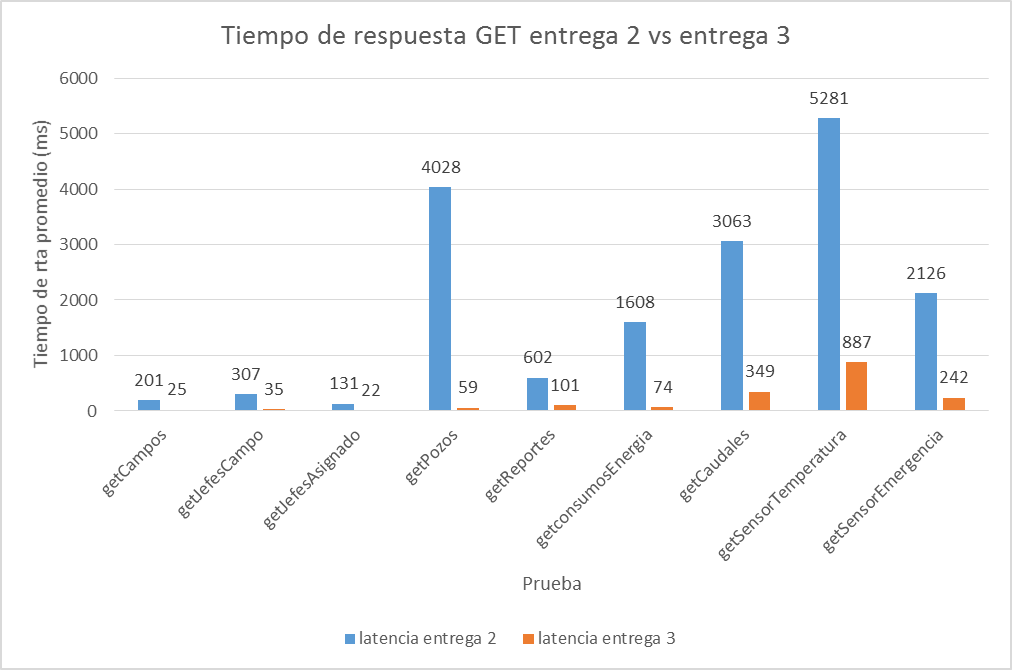
**Tabla de resumen 2**



Procedemos a comparar los tiempos de respuesta promedio de las funcionalidades presentes en ambas entregas para comprobar si una implementación es mejor que la otra.

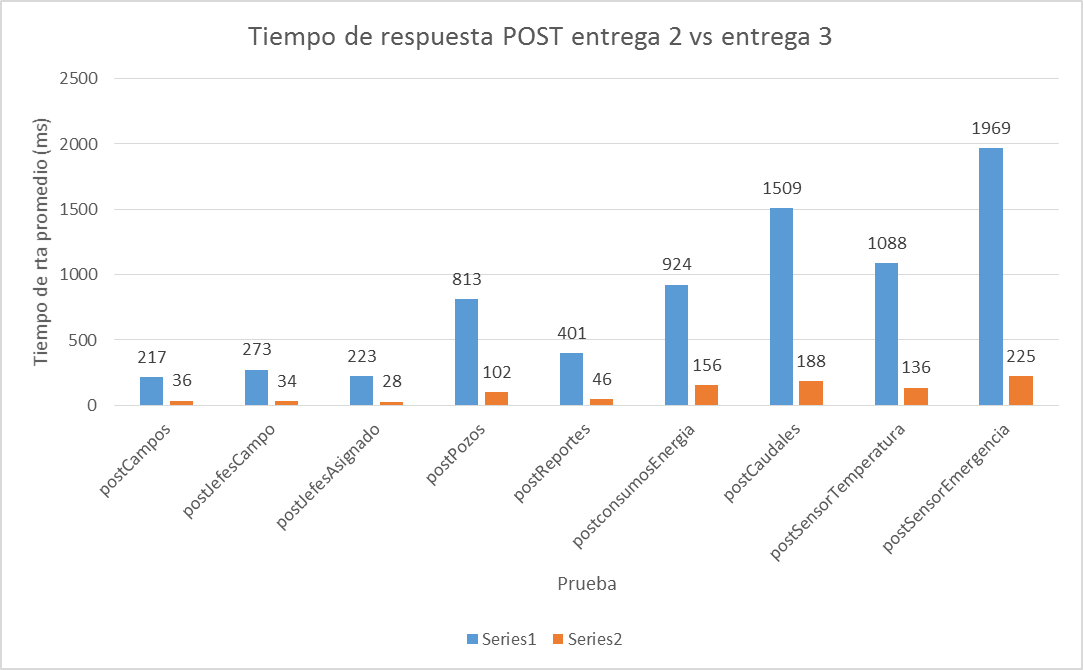
* Comparación del tiempo de respuesta para las pruebas GET.





* Comparación del tiempo de respuesta para las pruebas POST.



****

Tras realizar esta comparación el equipo de desarrollo se ve sorprendido al ver que los tiempos reportados en la entrega 3 pues se esperaba que esta implementación fuera más lenta al introducir elementos que consumían recursos de la aplicación. Se procedió, entonces a verificar las condiciones de ambas alternativas.

Es en este punto donde se logra entender el por qué los datos no concordaban con la lógica. Recordamos que esta implementación involucra la introducción de un balanceador de carga, el cual agiliza los procesos al distribuirlos, y que en esta iteración contábamos con 3 máquinas funcionando conjuntamente para cumplir con lo propuesto por este atributo de calidad. Así pues una distribución de las tareas y un aumento de un 200% en los recursos pudo llevar a una mejora en el desempeño.

Se concluyó, entonces, que al comparar los resultados de ambas iteraciones es evidente que la implementación de un balanceador de carga, que trae consigo un aumento de recursos en este caso, mejora el comportamiento de la aplicación en cuanto a su desempeño.

**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 se pueden satisfacer la cantidad de solicitudes posibles que fueron presentadas con la solución actual
* Se identificó que al comparar los resultados de ambas iteraciones es evidente que la implementación de un balanceador de carga, que trae consigo un aumento de recursos en este caso, mejora el comportamiento de la aplicación en cuanto a su desempeño.
* Sería interesante llevar a cabo pruebas de disponibilidad para ver si además de mejorar el desempeño de la aplicación, el balanceador de carga aumenta realmente la disponibilidad del sistema.