**Documentación de las pruebas Experimento 1, Entrega 2**

**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 1 y 2 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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

Ruta: /reportes

Parameters (JSON):

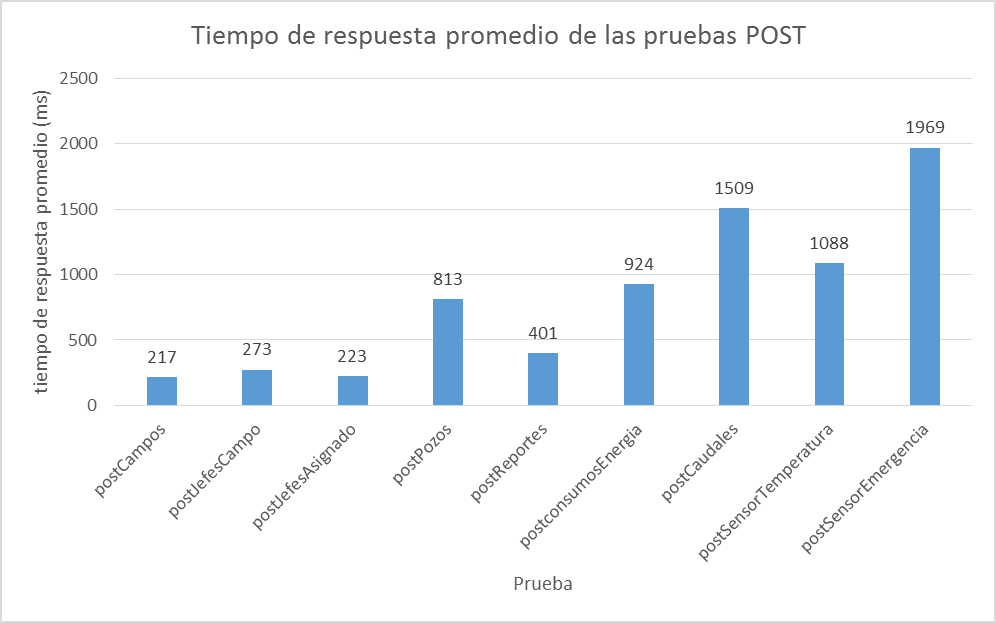
{ "descripcion": "hola"}

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

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: 9000

Ruta: /reportes

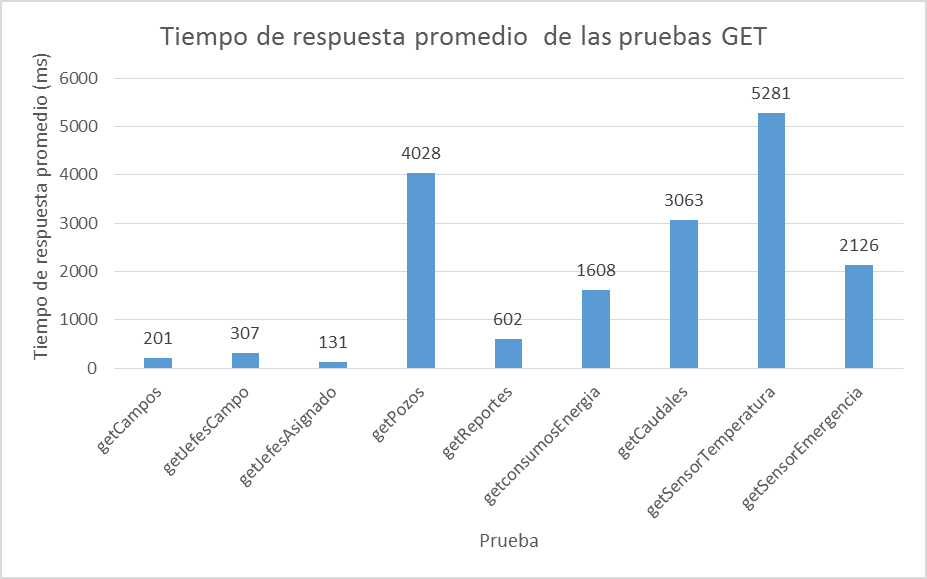
Parameters: vacío pues es un get

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**



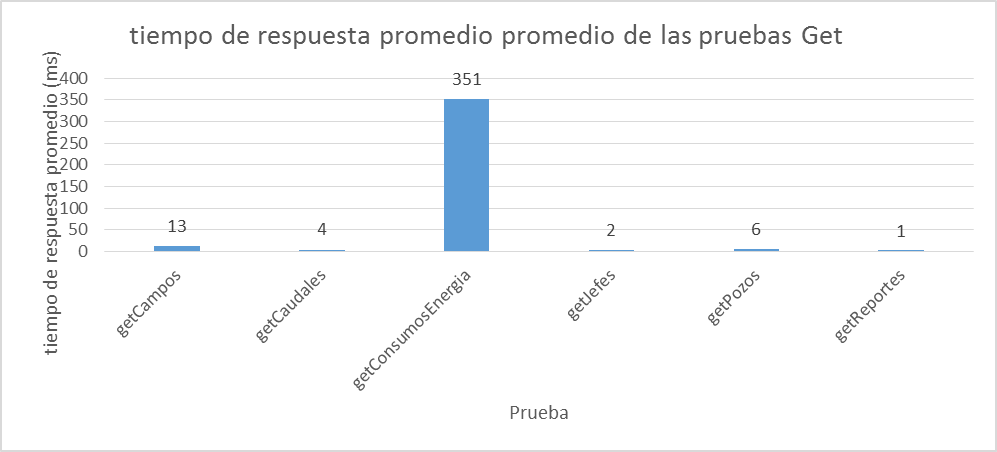
**COMPARACIÓN DE LOS RESULTADOS ENTRE ENTREGAS**

Para esta entrega mudamos el proyecto a PLAY, trabajando en IntelliJ IDEA por su fácil usabilidad en comparación al trabajo realizado con JEE en NetBeans.

A continuación se observan las tablas de resumen de la entrega 1 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 1**

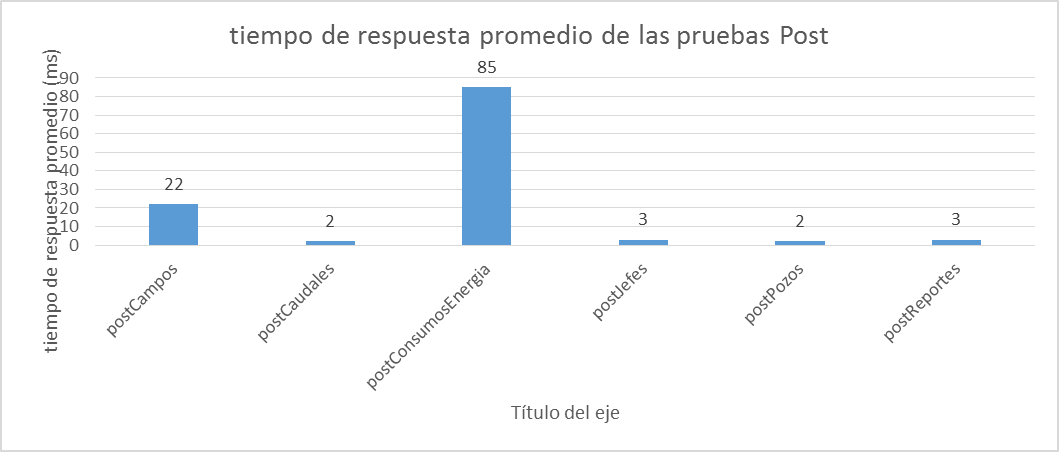
****

**Tabla de resumen1**



**POST**

**Tiempos de respuesta promedio de la aplicación a lo largo de la prueba 1**

****

**Tabla de resumen 1**

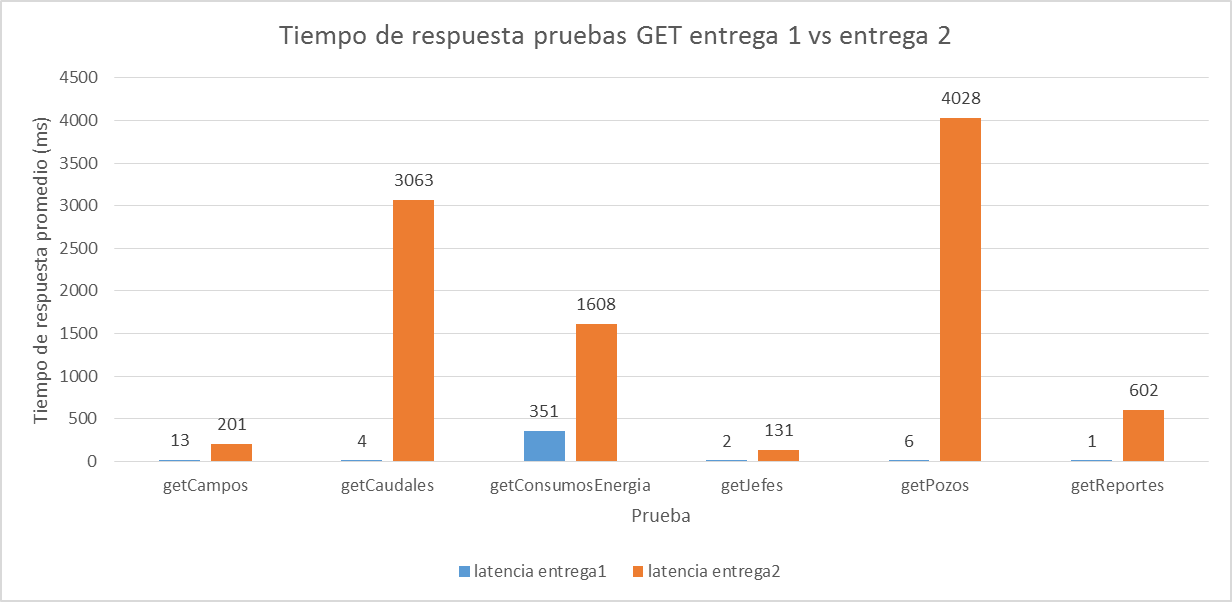


Teniendo en cuenta la construcción de las diversas pruebas para esta entrega, se evidencia inmediatamente que la cantidad de servicios ofrecidos es superior en este momento, y que ya no hay presencia de servicios que tengan un porcentaje de error mayor al 0%.

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

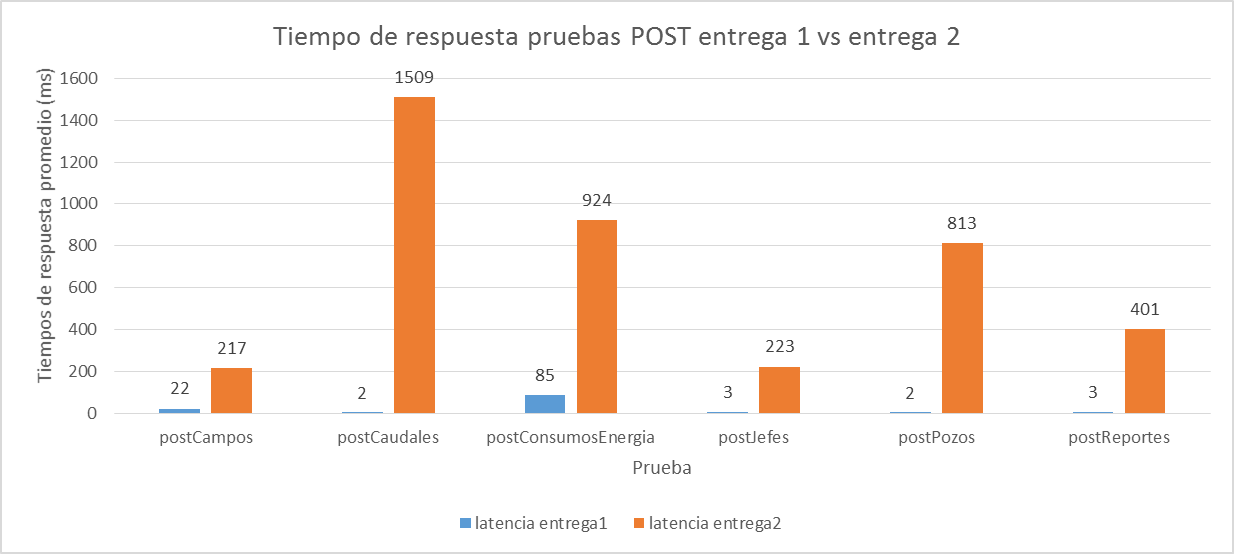
* Comparación del tiempo de respuesta para las pruebas GET.





* 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 1 son mucho mejores (al ser menores) que los de la presente entrega, a pesar de sentir que en esta implementación la aplicación corría de mejor manera. 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. Nos dimos cuenta que las pruebas de la primera entrega estaban sujetas a parámetros distintos. En la primer iteración todas las pruebas se realizaron con un “ramp up period” de 60 segundos y se revisó únicamente que los tiempos de respuesta fueran inferiores a 1 segundo. En esta entrega, todas las pruebas se realizaron con un “ramp up period” de 1 segundo, suponiendo, según lo indicado en el enunciado, que es posible que lleguen todas las peticiones de los sensores al mismo tiempo, en un lapso de 1 segundo, no de 1 minuto.

Analizando la situación es claro que los resultados van a ser muy alejados los unos de los otros pues 59 segundos extra para crear y enviar hilos de ejecución, y procesar dichas transacciones resultan en un desempeño mucho mejor, pues la demanda de recursos puede ser abastecida con mayor facilidad en un lapso de tiempo superior.

Se concluyó, entonces, que no es acertado comparar los resultados de ambas iteraciones para ver que metodología empleada es mejor ya que las pruebas se realizaron con el parámetro “ramp up period” distinto, generando diferencias significativas en los resultados.

**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 no es acertado comparar los resultados para ver que metodología empleada era mejor ya que las pruebas se realizaron con el parámetro “ramp up period” distinto, generando diferencias significatívas en los resultados.