

ISIS2503 ARQUITECTURA Y DISEÑO DE SOFTWARE

Joan David Torres Pinzon

Luis Andres Mesa

Juan Pablo Gonzalez

Felipe Martinez

Pre-experimentación

Problemática

OilCol S.A desea adquirir un software para el procesamiento y monitoreo automatizado de datos sobre sus pozos, campos y regiones de explotación de petróleo. Mediante la experimentación se explorarán las diferentes opciones de arquitecturas que satisfagan los atributos de calidad deseados, específicamente para este experimento se trabajara la arquitectura basada en Play-Framework.

Objetivo del experimento

Dado que una arquitectura de tipo REST es propensa a presentar problemas cuando hay una alta concurrencia de usuarios y pueden ocurrir errores de “DeathLock”, no va a satisfacer el atributo de calidad de “Desempeño”, adicionalmente un aplicación desarrollada mediante EJB, está diseñada para ejecutar un única tarea, donde los objetos serán de tipo stateless para facilitar el incremento de carga y hacer que las aplicaciones sean escalables. Asimismo, el servidor de aplicaciones se encarga de asignar los hilos de ejecución requeridos para ejecutar los procesos y la tarea asignada, esto hace que la aplicación esté fuertemente ligadas a sus servidores de aplicaciones(contenedores), por lo tanto el atributo de calidad correspondiente a la “Escalabilidad” se ve restringida a la forma que se despliegue el servidor

Por lo tanto la decisión para la construcción de la aplicación se realiza sobre Play Framework como solución de arquitectura reactiva que busca priorizar aspectos como la tolerancia a fallos, la disponibilidad y la elasticidad; aspectos primordiales para los atributos de calidad del experimento 1. Mediante el modelo de actores de Akka se busca hacer la ejecución de cientos de tareas en forma paralela, dado que son diseñados para ser muy ligeros, alrededor de 300 bytes por actores, lo que conlleva a calcular 2.5 millones de actores por cada GB en el heap de la Java Virtual Machine(JVM). En la arquitectura de Play toda petición que llega al enrutador es controlada por un componente interno llamado “Global Dispatcher”, que se encarga de desligar las peticiones de los clientes para utilizar múltiples Actores de Akka en el procesamiento de la petición, esto de forma asíncrona y no bloqueante,

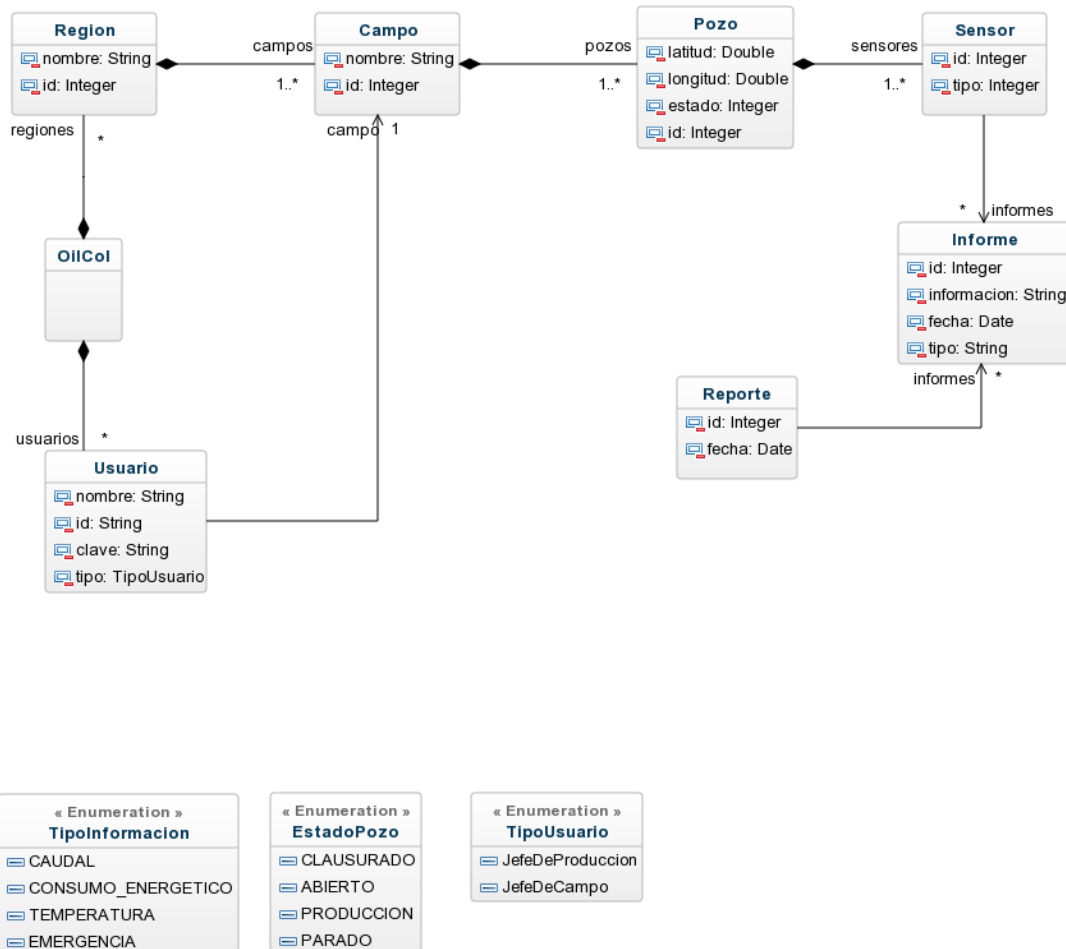
brinda como característica adicional la reutilización de recursos finalmente se utiliza una táctica de concurrencia asíncrona basado en eventos.

Descripción del experimento

Se someterá la arquitectura Play a pruebas de carga que resolverán escenarios de prueba específicos (sobre el Crud de las entidades) basados en el atributo de desempeño mediante un cluster de la aplicación Jmeter. También se evaluará el atributo de escalabilidad al responder a las tareas de forma eficiente distribuyendo los recursos disponibles, las peticiones se realizarán desde 3 máquinas virtuales también por un cluster en Jmeter.

Artefactos a construir

Dentro del paquete de modelos se crearán las entidades del negocio que corresponden al siguiente modelo de datos.



Adicionalmente en el paquete de Controllers se crearán los controladores encargados del CRUD y la persistencia sobre la base de datos de la lógica del negocio, haciendo uso de todas las implementaciones anteriores (modelo de actores, Dispatcher, y métodos asíncronos).

Se realizarán cambios en el archivo de configuración de la aplicación(application.conf) para independizar la base de datos del cliente consumidor de la aplicación, así en una máquina virtual estarán las bases de datos y en las demás existirá el hilo de la aplicación.

Recursos de la experimentación

Las máquinas siguen las siguientes configuraciones:

- Disponer de Java Development Kit versión 8 (JDK 8).
- Tener la configuración de la variable JAVA_HOME en las variables de entorno.
- Disponer de un entorno de desarrollo integrado (IDE) IntelliJ Community Edition 2016.1.2 con el plugin de Scala.
- Tener instalado y configurado previamente una base de datos PostgreSQL. (sobre la máquina destinada a ser el servidor de datos)
- Tener instalado y configurado JMeter versión 3.0 (tanto en los esclavos como en el manejador general)

Resultados esperados

La aplicación debe cumplir con los escenarios mínimos de calidad enunciados en las pruebas, al correr todas las peticiones del CRUD de cada entidad eficazmente.

Duración y etapas

En un plazo de semana y media se realiza el experimento parcial, que buscará dar respuesta a:

- ❖ El sistema es capaz de recibir información de cada uno de los siguientes sensores
 - Sensor de temperatura de la bomba
 - Sensor de consumo energético
 - Sensor de cantidad de barriles de crudo
 - Sensor de emergencia
- ❖ El sistema consulta la información de pozos, campos y zonas geográficas para generación de reportes utilizando filtros temporales
- ❖ El sistema permite cambiar los estados del pozo
- ❖ El sistema maneja la información de los pozos, campos y zonas geográficas (CRUD)

- ❖ El sistema soporta la recepción de información desde los 4800 sensores en una ventana de tiempo de 1 segundo

Experimentación

Se realizarán para el atributo de calidad Desempeño los siguientes escenarios.

Identificador	Tipo	Prioridad
1	Desempeño	Alta
Fuente		
Sensores		
Estímulo		
Carga de datos sobre el sistema		
Ambiente		
Normal		
Medida Esperada		
Se recibe información desde los 4800 sensores en una ventana de tiempo de 1 segundo		

Identificador	Tipo	Prioridad
1	Desempeño	Alta
Fuente		
Sensores		
Estímulo		
Carga de datos sobre el sistema		
Ambiente		
Sobrecargado		
Medida Esperada		

El sistema debe procesar la información enviada(9800 solicitudes en una ventana de tiempo de 1s) desde los sistemas con un error menor al 5%

Para la configuración del Jmeter **maestro** se realizaron las siguientes configuraciones:

```
#If set and the directory does not exist, it will be created.
#jmeter.gui.action.save.backup_directory=

#Set the maximum time (in hours) that backup files should be
preserved since the save time.
#By default no expiration time is set which means we keep
backups for ever.
#jmeter.gui.action.save.keep_backup_max_hours=0

#Set the maximum number of backup files that should be
preserved. By default 10 backups will be preserved.
#Setting this to zero will cause the backups to not being
deleted (unless keep_backup_max_hours is set to a non zero
value)
#jmeter.gui.action.save.keep_backup_max_count=10

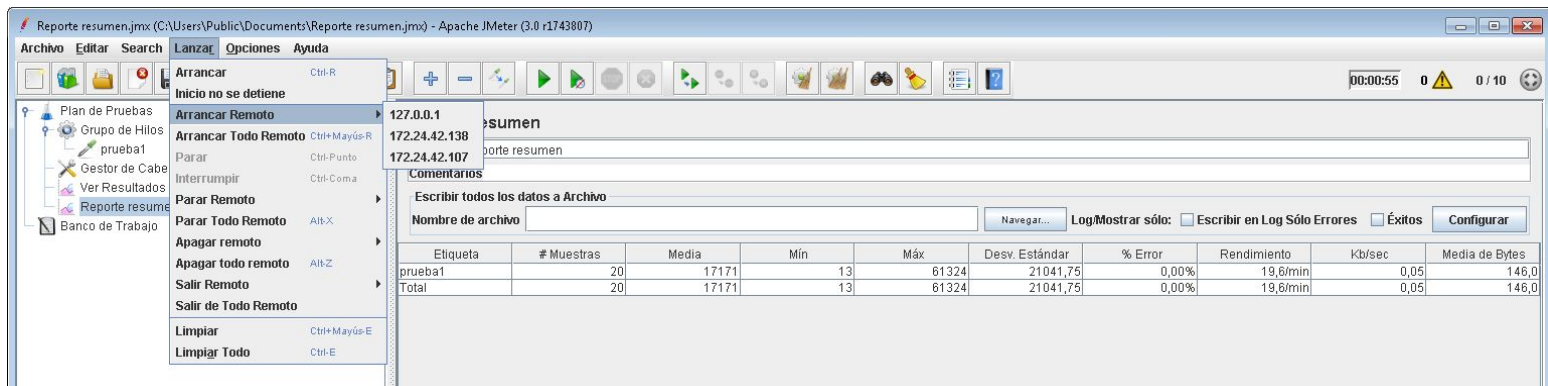
#-----
# Remote hosts and RMI configuration
#-----

# Remote Hosts - comma delimited
remote_hosts=127.0.0.1,172.24.42.138,172.24.42.107
#remote_hosts=localhost:1099,localhost:2010

# RMI port to be used by the server (must start rmiregistry
with same port)
#server_port=1099

# To change the port to (say) 1234:
# On the server(s)
# - set server_port=1234
# - start rmiregistry with port 1234
# On Windows this can be done by:
# SET SERVER_PORT=1234
# JMETER-SERVER
#
# On Unix:
# SERVER_PORT=1234 jmeter-server
#
# On the client:
# - set remote_hosts=server:1234

# Parameter that controls the RMI port used by the
RemoteSampleListenerImpl (The Controller)
# Default value is 0 which means port is randomly assigned
# You may need to open Firewall port on the Controller machine
```



Reporte resumen.jmx (C:\Users\Public\Documents\Reporte resumen.jmx) - Apache JMeter (3.0 r1743807)

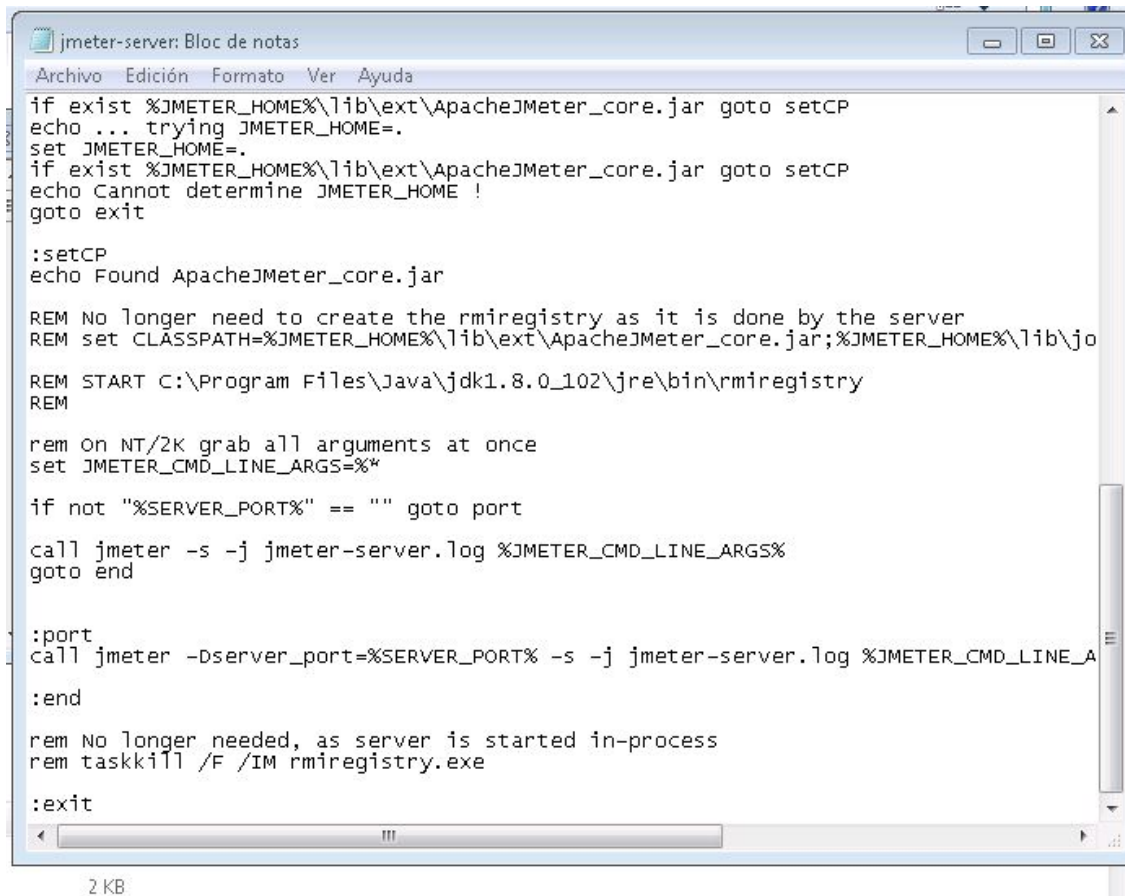
Archivo Editar Search Lanzar Opciones Ayuda

Arrancar Ctrl-R
Inicio no se detiene
Arrancar Remoto
Arrancar Todo Remoto Ctrl+Mayúsc+R
Parar Ctrl-Punto
Interrumpir Ctrl-Coma
Parar Remoto
Parar Todo Remoto Alt+X
Apagar remoto
Apagar todo remoto Alt+Z
Salir Remoto
Salir de Todo Remoto
Limpiar Ctrl+Mayúsc+E
Limpiar Todo Ctrl+E

127.0.0.1
172.24.42.138
172.24.42.107
Comentarios
Escribir todos los datos a Archivo
Nombre de archivo
Navegar... Log/Mostrar sólo: ☐ Escribir en Log ☐ Sólo Errores ☐ Éxitos Configurar

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
prueba1	20	17171	13	61324	21041,75	0,00%	19,6/min	0,05	146,0
Total	20	17171	13	61324	21041,75	0,00%	19,6/min	0,05	146,0

Para cada **esclavo** se realizaron las siguientes configuraciones, y mediante la consola se comprueba que la conexión con el maestro fue exitosa y se realizó la prueba.



jmeter-server: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
if exist %JMeter_HOME%\lib\ext\ApacheJMeter_core.jar goto setCP
echo ... trying JMeter_HOME=.
set JMeter_HOME=.
if exist %JMeter_HOME%\lib\ext\ApacheJMeter_core.jar goto setCP
echo Cannot determine JMeter_HOME !
goto exit

:setCP
echo Found ApacheJMeter_core.jar

REM No longer need to create the rmiregistry as it is done by the server
REM set CLASSPATH=%JMeter_HOME%\lib\ext\ApacheJMeter_core.jar;%JMeter_HOME%\lib\jo

REM START C:\Program Files\Java\jdk1.8.0_102\jre\bin\rmiregistry
REM

rem On NT/2K grab all arguments at once
set JMeter_CMD_LINE_ARGS=%*

if not "%SERVER_PORT%" == "" goto port

call jmeter -s -j jmeter-server.log %JMeter_CMD_LINE_ARGS%
goto end

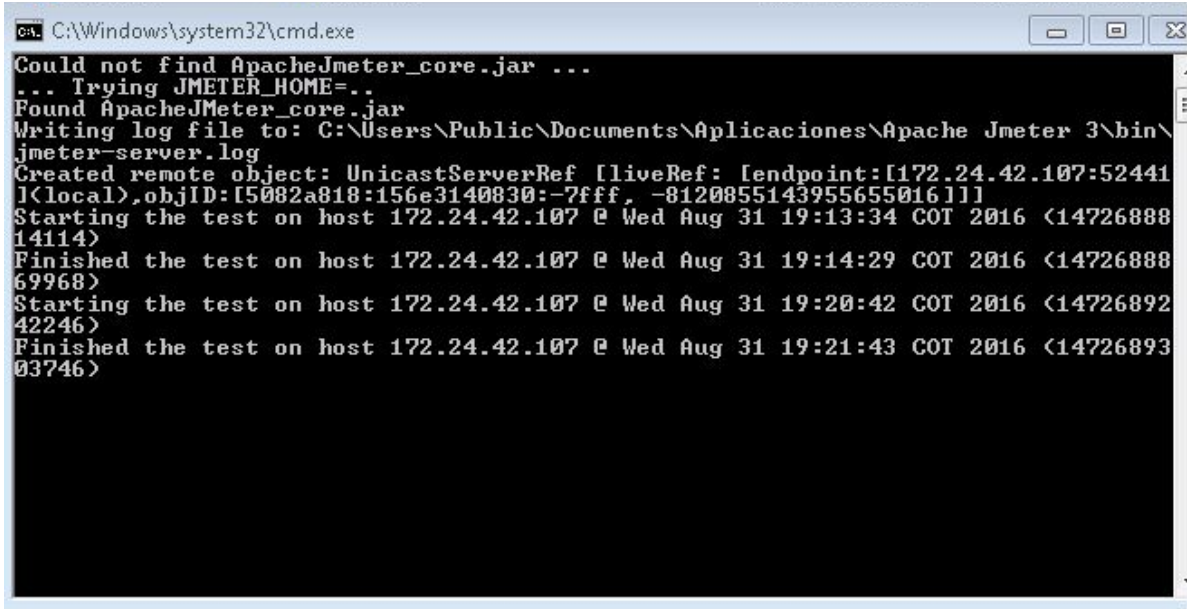
:port
call jmeter -Dserver_port=%SERVER_PORT% -s -j jmeter-server.log %JMeter_CMD_LINE_A

:end

rem No longer needed, as server is started in-process
rem taskkill /F /IM rmiregistry.exe

:exit
```

2 KB



```
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: C:\Users\Public\Documents\Aplicaciones\Apache Jmeter 3\bin\
jmeter-server.log
Created remote object: UnicastServerRef [liveRef: [endpoint:[172.24.42.107:52441
l<local>,objID:[5082a818:156e3140830:-7fff, -81208551439556550161]]
Starting the test on host 172.24.42.107 @ Wed Aug 31 19:13:34 COT 2016 <14726888
14114>
Finished the test on host 172.24.42.107 @ Wed Aug 31 19:14:29 COT 2016 <14726888
69968>
Starting the test on host 172.24.42.107 @ Wed Aug 31 19:20:42 COT 2016 <14726892
42246>
Finished the test on host 172.24.42.107 @ Wed Aug 31 19:21:43 COT 2016 <14726893
03746>
```

Post-experimentación

Resultados obtenidos:

Duración real

Las imágenes que se ven a continuación muestran el proceso de prueba y algunos resultados obtenidos en jmeter. Posteriormente se podrá observar un tabla de datos con su correspondiente gráfica que permitirá analizar la información de forma ordenada y visual.

umen.jmx - Apache JMeter (3.0 r1743807)

00:00:02 0 0 / 1600

Petición HTTP

Nombre: prueba1

Comentarios

Basic Advanced

Servidor Web

Nombre de Servidor o IP: localhost Puerto: 9000 Timeout (milisegundos) Conexión: Respuesta:

Petición HTTP

Implementación HTTP: Java Protocolo: Método: POST Codificación del contenido:

Ruta: /informe

☐ Redirigir Automáticamente ☒ Seguir Redirecciones ☒ Utilizar KeepAlive ☐ Usar 'multipart/form-data' para HTTP POST ☐ Cabeceras compatibles con navegadores

Parameters Body Data Files Upload

Enviar Parámetros Con la Petición:

Nombre:	Valor	¿Codificar?	¿Incluir Equals?
{	"dato": 17673647, "emergencia": false, "tipo": "control" }	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Detail Añadir Add from Clipboard Borrar Up Down

Servidor Proxy

Nombre de Servidor o IP: Puerto: Nombre de Usuario Contraseña

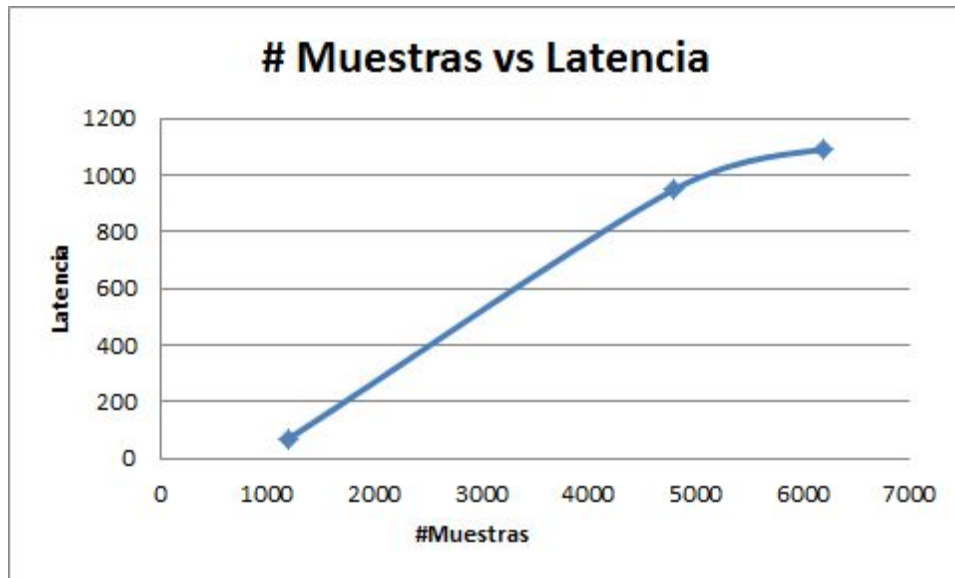
Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
prueba1	4800	3978	9	21889	5878,09	0,00%	64,9/sec	11,45	180,8
Total	4800	3978	9	21889	5878,09	0,00%	64,9/sec	11,45	180,8

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Petición HTTP	4800	2339	38	8249	2062,63	8,98%	305,5/sec	99,94	335,0
Total	4800	2339	38	8249	2062,63	8,98%	305,5/sec	99,94	335,0

Artefactos construidos

- Cada una de las clases se construyó efectivamente en la aplicación, los cambios respecto a la planeación se dieron con respecto a la decisión de implementar la persistencia no mediante una clase "Mock" para cada entidad, sino mediante una base de datos configurada en postgresql 9.4 en una de las máquinas virtuales.
- Cada aplicación se conecta a la base de datos como el mismo usuario para realizar las consultas y la persistencia de los datos, cabe resaltar que este es un proceso asíncrono haciendo uso de los FutureTask.

Análisis



La gráfica que se muestra expone el número de muestras y la latencia que tuvo, donde se ve un crecimiento lineal en el comienzo y finaliza de forma tal que a partir de cierta cantidad de datos esta latencia puede llegar a ser constante. Lo que se hizo fue analizar el resultado de el número de registros que hace un sensor en un segundo. Lo que se busca es que el sistema responda a 4800 peticiones en un segundo, sin embargo esto no se logró ya que el tiempo fue mayor y en las últimas pruebas se encontró porcentajes de error. Los resultados obtenidos pueden verse afectados a causa de la latencia que hay en el envío de peticiones a la base de datos y el tiempo de respuesta que esta se demora.



La gráfica anterior representa pruebas realizadas para estudiar el funcionamiento del método post de la entidad usuario respecto a el tiempo medio de respuesta y el número de muestras, estas se realizaron en una ventana de tiempo de 60 segundos, se puede observar el excelente comportamiento del software hasta antes de los 4000 hilos ejecutados en JMeter debido a su a que su tiempo de respuesta máximo fue de 282 milisegundos y así mismo los errores solo se presentan al sobrepasar las 4000 solicitudes.

Conclusiones

Después de analizar los resultados obtenidos a lo largo de la experimentación, se llegó a la conclusión que la arquitectura planteada en esta primera etapa, no logra responder de la manera mas optima a las exigencias del proyecto, debido al hecho de que se presentan errores bajo condiciones normales, los cuales no son aceptables. Por otro lado, bajo un ambiente de sobrecarga el sistema tampoco se comportó dentro de lo establecido antes de la experimentación y presentó porcentajes de error superiores al 5%. Durante esta primera etapa, se lograron identificar falencias en el sistema que abren nuevos retos en el proceso de lograr una arquitectura que satisfaga todos los requerimientos planteados en el proyecto. Se reconoce la necesidad de optimizar la implementación actual para lograr un mejor uso de los recursos y así, un mejor desempeño y escalabilidad.

Repositorio de la aplicación

<https://github.com/f-martinez11/Oilcol>