Universidad Politécnica de Valencia
E.T.S. Ingeniería Informática
Máster en Ingeniería Informática
MUIInf

Curso 2019/2020

Sistemas y Aplicaciones Distribuidas

Memoria Prácticas de Laboratorio

Pérez Martín, Ismael

ÍNDICE GENERAL

1

INFORMACIÓN IMPORTANTE

A continuación se detallan una serie de recursos o información necesarios para la evaluación y compensión de esta memoria.

1.1 REPOSITORIO

El código desarrollado durante todas las prácticas de laboratorio se ha gestionado mediante el sistema de control de versiones Git[1].
Este repositorio se ha almacenado en el sistema de almacenamiento en la nube para repositorios GitHub[2], y se encuentra disponible en el siguiente enlace:

`https://github.com/ismpere/SAD`

1.2 ORGANIZACIÓN DEL REPOSITORIO

El repositorio está organizado en carpetas, por lo cual para poder acceder a cada uno de los seminarios basta con:

- Acceder a la URL del repositorio
- Acceder a la carpeta Labs
- Acceder a la carpeta del laboratorio correspondiente

Cada uno de los laboratorios cuenta con un README.md el cual presenta información importante del laboratorio así como los pasos necesarios para su instalación y/o ejecución.

2

DESCRIPCIÓN DEL TRABAJO DE LABORATORIO

2.1 TRABAJO REALIZADO Y POR HACER

En este apartado se detalla el trabajo completado durante las prácticas de laboratorio.

2.1.1 *Trabajo realizado*

Durante el periodo de prácticas se han completado todos los laboratorios disponibles:

- **1.1**: Introducción a Javascript y Node.js
- **1.2**: Desarrollo de un Foro. Cliente/Servidor
- **1.3**: Desarrollo de un Foro. Servidor de datos y cliente
- **2.1**: Backend distribuido. ZeroMQ
- **2.2**: Backend distribuido. Grupo de servidores
- **2.3**: Backend distribuido. Consistencia

En estos laboratorios se ha implementado el 100 % de la funcionalidad requerida por el laboratorio, contando en su versión final con importantes funcionalidades como:

- Independencia del servidor de datos y el foro
- Intercambio de mensajes mediante protocolo Rep/Req y sockets ZeroMQ[3]
- Intercambio de mensajes mediante protocolo Pub/Sub y sockets ZeroMQ
- Propagación de mensajes a diferentes servidores de datos y foros
- Pruebas de consistencia contra uno o un grupo de servidores

2.1.2 *Trabajo por hacer*

Se han realizado las prácticas al completo por lo que no hay ninguna funcionalidad obligatoria restante por implementar o modificar.

2.2 FALLOS CONOCIDOS DEL SOFTWARE

Los fallos detectados son los siguientes:

- **Procesos en segundo plano Linux**:
  Al parar los procesos en Linux, algunas veces el proceso se queda ejecutándose en segundo plano, por lo que es necesario escanear que procesos hay en cada puertos y parar dichos procesos antes de volver a ejecutar el laboratorio.
  Esto puede ser propiciado por una mala detección del evento *close* en los sockets ZeroMQ, ya que solo ocurre al utilizar estos.

- **npm install no instala zeromq**:
  A pesar de incluir distintas versiones de zeromq en el package.json, no se ha conseguido instalar zeromq a menos que se instalara manualmente mediante *npm install zeromq*.

2.3 POSIBLES MEJORAS

En este apartado se detallan las mejora implementadas o deseables para una futura implementación de cara a la mejora del sistema.

2.3.1 *Mejoras realizadas*

A continuación se detallan las posibles mejoras del software realizadas:

- **Mejora del log**:
  Se ha añadido un log más claro y abundante tanto en el servidor como el foro para poder tener una mejor detección de errores.

- **Creación de scripts de despliegue y prueba**:
  Se han creado estos archivos (runner.js y tester.js) para agrupar diferentes comandos necesarios para iniciar los servidores.
  Estos scripts ahorran mucho tiempo en el despliegue y prueba de los elementos desarrollados, sobre todo en el caso de los grupos de servidores,

2.3.2 *Mejoras sin realizar*

A continuación se detallan las posibles mejoras del software propuestas:

- **Remplazo de la función retardo**:
  La función retardo hace uso de un while que bloquea el programa durante su ejecución, lo cual no es para nada eficiente de cara a la ejecución del resto del programa.
  La mejora propuesta es hacer uso de la función *system.timeout* que retrasa la ejecución de la instrucción de manera asíncrona pero el programa sigue ejecutándose de manera habitual.

- **Implementación total mediante Pub/Sub**:
  Implementar el resto de funcionalidades necesarias mediante Pub/Sub (excluyendo funciones como los getter), daría al sistema una visión mucho más distribuida que afectaría por igual al listado de servidores.

- **Uso de una base de datos**:
  Al usarse una lista local los cambios son volátiles, lo cual resulta poco beneficioso para un despliegue más realista de la aplicación y con unos cambios permanentes.

- **Crear servicio de registro de servidores**:
  El tener que introducir manualmente el listado de servidores del sistema es algo para nada seguro e ineficaz.
  La solución propuesta es crear un registry en el cual los servidores se den de alta una vez se inicien para tener una lista de servidores los cuales publican o se suscriben a cambios.

- **Comprobación de la marca temporal de los mensaje**:
  Desechar los mensajes que se reciban fuera de su marca temporal u ordenar los mensajes al suceder uno de estos casos sería una buena idea para mejorar la consistencia del sistema.

2.4 PRUEBAS DE VERIFICACIÓN DEL SISTEMA

A continuación se detallan las pruebas realizadas para la verificación del buen funcionamiento del sistema:

1. **Despliegue del foro**:

   a) Ejecutar node dmserver.js

   b) Ejecutar node forum.js

   c) Acceder a `http://localhost:10000/`

   Resultado esperado: El foro está desplegado en esa url

2. **Login**:

a) **Login correcto**

1) Desplegar el foro (1)
2) Introducir credenciales válidas
3) Pulsar enter

Resultado esperado: El usuario entra al sistema y se desbloquea la barra de mensajes

b) **Login incorrecto**

1) Desplegar el foro (1)
2) Introducir credenciales no válidas
3) Pulsar enter

Resultado esperado: El usuario no entra al sistema y no se desbloquea la barra de mensajes

3. **Añadir mensaje público**:

a) Hacer login válido (2.a)
b) Escribir un mensaje en la barra de mensajes
c) Pulsar enter

Resultado esperado: El mensaje se añade en el foro y en todos los foros restantes

4. **Añadir mensaje privado**:

a) Hacer login válido (2.a)
b) Seleccionar un usuario
c) Escribir un mensaje en la barra de mensajes
d) Pulsar enter

Resultado esperado: El mensaje se añade en el foro actual y solo se muestra en la conversación privada
Este test es suficiente para probar el resto de funcionalidades no implementadas mediante protocolo Pub/Sub.

5. **Despliegue de grupo de servidores**:

a) Ejecutar el runner (despliegue conjunto)
b) Acceder a `http://localhost:10000/`, hacer login y enviar un mensaje
c) Acceder a `http://localhost:10002/`, hacer login y enviar un mensaje
d) Acceder a `http://localhost:10001/` y observar la barra de mensajes

Resultado esperado: Todos los foros tienen la misma lista de mensajes públicos

6. **Almacenamiento de mensajes**:

   a) Ejecutar el runner (despliegue conjunto)

   b) Acceder a `http://localhost:10000/`, hacer login y enviar un mensaje

   c) Recargar la página principal de los foros

   Resultado esperado: Todos los foros tienen la lista de mensajes junto con el último mensaje enviado.

7. **Envio de mensajes individual por foro**:

   a) Ejecutar el runner (despliegue conjunto)

   b) Acceder a `http://localhost:10000/`

   c) Abrir otra ventana y acceder a `http://localhost:10000/`

   d) Enviar un mensaje desde cualquiera de las dos ventanas

   Resultado esperado: El mensaje se ha añadido una sola vez y todos los foros tienen la lista de mensajes junto con el último mensaje enviado una sola vez.

8. **Funcionamiento individual del servidor de datos**:

   a) Ejecutar el runner (despliegue conjunto)

   b) Acceder a `http://localhost:10000/`, hacer login y enviar un mensaje

   c) Acceder a `http://localhost:10001/`

   Resultado esperado: La lista de mensajes mostrada en `http://localhost:10001/` es la misma que la que aparece en `http://localhost:10000/`.

2.5 PRUEBA CONVENCIONAL DE SOFTWARE

Probar la corrección del software. Para ello lanzaremos 3 servidores de datos *dmserver*. Estarán conectados entre ellos mediante ZeroMQ. Lanzaremos a su vez 3 servidores web *forum*. Cada uno de ellos conectado a un servidor de datos diferente. Lanzaremos 3 navegadores web. Cada uno de ellos conectado a un servidor web diferente. Posteando mensajes en cada navegador debemos observar que los mensajes se actualizan en los diferentes navegadores.

Esta prueba se realiza mediante los siguiente pasos detallados en el README.md del laboratorio 2.3. Hay dos maneras para realizar dicha prueba:

1. Ejecutar runner.js (despliega los tres servidores de datos y los tres foros)

```
node runner.js
```

2. Ejecutar tester.js (ejecuta runner.js)

```
node tester.js base
```

A continuación, se procede a probar los foros desplegados en:

- http://localhost:10000/
- http://localhost:10001/
- http://localhost:10002/

El foro se comporta de manera esperada y los mensajes se propagan adecuadamente por el resto de servidores y foros.

2.6 PRUEBAS DE CONSISTENCIA

A continuación se detalla el procedimiento y motivo de cada una de las pruebas de consistencia realizadas:

2.6.1 *Función retardo*

Esta función proporcionada en el seminario se ha implementado en el archivo dmserver.js. El servidor de datos tiene ahora un argumentos a mayores el cual se introduce de la manera -r(tiempo en milisegundos).
Este retardo se usa inmediatamente antes de publicar los mensajes públicos en el servidor de datos, quedando el servidor a la espera de que termine el retraso para efectuar la publicación mediante sockets ZeroMQ.

2.6.2 *Consistencia FIFO*

Sin retardo

Posteamos dos mensajes de forma consecutiva sobre el mismo *dmserver* y debemos observar cómo los dos mensajes llegan en el mismo orden a los navegadores web conectados.

En esta prueba se despliegan dmserver y forum en las rutas por defecto y se publican los mensaje mediante dmclient. Para ejecutar esta prueba se utiliza el tester desarrollado:

```
node tester.js 1.1
```

A continuación, se observa el resultado en el foro desplegado en http://localhost:10000/.

Resultado:

Se puede observar que los mensajes en algunos casos llegan desordenados, llegando antes el mensaje 2 que el mensaje 1, por lo que no se puede garantizar una consistencia FIFO en el sistema.

Con retardo

Repetimos el proceso añadiendo un *retardo* en el *dmserver*, justo antes de que éste publique sus cambios al servidor web.

Como se ha mencionado anteriormente, esta función se localiza justo antes de la publicación de los mensajes públicos.

En esta prueba se despliegan dmserver y forum en las rutas por defecto pero esta vez con un retraso de 2000 ms en el servidor de datos. Por último se publican los mensaje mediante dmclient. Para ejecutar esta prueba se utiliza el tester desarrollado:

```
node tester.js 1.2
```

A continuación, se observa el resultado en el foro desplegado en http://localhost:10000/.

Resultado:

Al igual que en el caso anterior, se puede observar que los mensajes en algunos casos llegan desordenados, por lo que no se puede garantizar una consistencia FIFO en el sistema.
A partir de un retardo en la publicación del mensaje de 3 segundos, algunos mensajes empiezan a perderse, aumentando esta posibilidad cuando mayor sea el retardo.

2.6.3 *Consistencia causal*

Prueba de funcionamiento

Supongamos los 3 *dmserver* llamados S1, S2 y S3. Posteamos un mensaje sobre S1. A continuación desde otro terminal, consultamos la lista de mensajes en el servidor S2, y acto seguido posteamos un nuevo mensaje sobre S2. Observamos qué mensajes aparecen en los navegadores.

En esta prueba se comprueba el correcto funcionamiento del sistema, y mencionado anteriormente en las pruebas del sistema realizadas, el procedimiento es el siguiente:

Desplegar los servidores mediante el runner desarrollado:

```
node runner.js
```

A continuación, realizan las pruebas detalladas en los foros desplegados en:

- http://localhost:10000/
- http://localhost:10001/
- http://localhost:10002/

Resultado:
Siguiendo el correcto funcionamiento del sistema, sin ningún retraso añadido, en los navegadores los mensajes aparecen ordenador por el orden en el que los hemos enviado, siguiendo una consistencia FIFO.

Provocación de fallo

Diseñamos una prueba para forzar el fallo en la consistencia causal, mediante *retardos* hemos de lograr observar que el navegador conectado a S3, muestre el mensaje enviado a S2, antes que el mensaje inicial.

Para diseñar esta prueba se han seguido los siguientes pasos:

- Se despliega el servidor 1 con un retardo de 1000ms
- Se despliega el resto de componentes del grupo de servidores
- Se ejecuta un retardo de 1500ms para esperar al inicio de los servidores de datos y foros
- Se ejecuta dmclient con una petición de publicación de mensaje público en S1
- Se ejecuta dmclient con una petición de publicación de mensaje público en S2

Este test ya está implementado en el tester desarrollado y se ejecuta de la siguiente manera:

```
node tester.js 2
```

A continuación, realizan las pruebas detalladas en los foros desplegados en:

- http://localhost:10000/
- http://localhost:10001/
- http://localhost:10002/

Resultado:
El segundo mensaje con contenido *Test dmClient 2* aparece antes que el primer mensaje con contenido *Test dmClient 1*. Esto se produce ya que, al tener el primer servidor un retraso en la publicación del mensaje, el mensaje publicado por el servidor 2 se envía antes y es recibido con antelación por el resto de servidores.
Esto muestra que el sistema si sufre algún retraso tampoco cumple con la consistencia causal.

2.6.4 *Consistencia secuencial*

Diseñar una prueba que muestre si nuestro sistema cumple o no la consistencia secuencial.

Para diseñar esta prueba se han seguido los siguientes pasos:

- Se inicia un servidor de datos y un foro en las direcciones por defecto
- Se ejecuta un bucle 10 veces encargado de:
  1. Publicar un mensaje con contenido *Test dmClient 1* mediante dmclient.
  2. Publicar un mensaje con contenido *Test dmClient 2* mediante dmclient.

Este test ya está implementado en el tester desarrollado y se ejecuta de la siguiente manera:

```
node tester.js 3
```

Al finalizar esta prueba se accede a `http://localhost:10000/` para observar los resultados. Para cumplir la consistencia secuencial todos los mensajes deberían seguir el orden 1, 2, 1, 2...

Resultado:

Se pude observar que en más de una ocasión los mensajes no han llegado de la misma manera, por lo que ejecutando la misma secuencia 10 veces seguidas no obtenemos el mismo resultado. Con esto podemos concretar que tampoco cumple la consistencia secuencial.

2.7 CONSISTENCIA DEL SISTEMA

Tras realizar las pruebas de consistencia del sistema, se puede deducir que el sistema **no cumple ninguna de las consistencias**:

- **Consistencia FIFO**:
  Como hemos visto en las pruebas, los mensajes pueden llegar en un orden diferente al que se enviaron por lo que no se cumple esta consistencia.

- **Consistencia causal**:
  Como se ha detallado en las pruebas, al añadir un retraso y debido a que no hay control del timestamp de los mensajes, los mensajes pueden recibirse desordenados, por lo que tampoco se cumple la consistencia causal.

- **Consistencia secuencial**:
  Por último, observando el último test de consistencia, se observa que las operaciones ejecutadas en el mismo orden no tienen el mismo resultado, por lo que tampoco se cumple la consistencia secuencial.

BIBLIOGRAFÍA

[1] GIT,

Consultado el 19/01/2020, Disponible en:

https://git-scm.com/

[2] GITHUB,

Consultado el 19/01/2020, Disponible en:

https://github.com/

[3] ZEROMQ,

Consultado el 19/01/2020, Disponible en:

https://zeromq.org/