Práctica 4: Servicio de gestión de vistas

Javier Fuster Trallero Javier Herrer Torres

Noviembre 2020

1. Introducción

En esta práctica se introduce la tolerancia a fallos para nodos distribuidos con estado. El objetivo es construir un servicio de almacenamiento clave/valor tolerante a fallos utilizando replicación Primario/Copia en memoria RAM. En esta práctica se diseñará e implementará únicamente el servicio de gestión de vistas.

2. Gestor de vistas

El gestor de vistas podrá recibir los siguientes tipos de mensaje:

- `MsgLatido` enviado por los servidores para notificar que «siguen vivos».
- `MsgPeticionVistaValida` usado para depuración.
- `MsgPeticionPrimario` usado para depuración.
- `MsgTickInterno` recibido cada 50 milisegundos para comprobar si los servidores siguen vivos.
- `MsgFin` causará la finalización del proceso.

2.1. Tratamiento de un latido

Existen tres tipos de latidos dependiendo de su argumento:

- **Argumento 0**: se produce cuando una máquina contacta inicialmente (vista 0), o cuando ha sufrido una recaída. Si se recibe al inicio, se posicionará al remitente como primario o copia (creando una nueva vista). Si se recibe posteriormente, se comprobará si se trata de un nuevo cliente y, en caso de no ser uno nuevo, se tratará como una caída de uno de los servidores (ver 2.3). Por último, se añadirá al servidor a la lista de servidores en espera.
- **Confirmar vista**: el primario confirmará la vista tentativa cuando exista inconsistencia (vista válida y tentativa diferentes). Enviará como argumento del latido el número de vista tentativa.

- **Argumento -1**: situación excepcional que se produce al inicio, cuando el primario no desea ni confirmar la vista ni notificar una caída.

Finalmente, se comprobará si es necesario promocionar un servidor en espera a copia. Se reiniciará el contador de retrasos asociado a ese servidor. Y, por último, se le contestará con la vista tentativa.

2.2. Procesar la situación de las réplicas

Se incrementará en una unidad el contador de retrasos asociado a cada servidor y, si alguno de ellos ha llegado al máximo de latidos fallidos permitidos, se tratará como una caída del servidor (ver 2.3).

2.3. Procesar un servidor caído

Cuando se notifica un servidor como caído, si no ha sido rearrancado será eliminado de la lista de servidores en espera. Posteriormente, se comprobará si el servidor caído es primario o copia. En caso de caer el primario y no estar en un estado consistente, se producirá con error fatal. Si la vista valida y tentativa son iguales, se promocionará a la copia. En caso de caer la copia, se promocionará un servidor en espera a copia.

3. Pruebas realizadas

El diagrama de secuencia de la figura 1 muestra la ejecución de las diferentes pruebas.

3.1. No debería haber primario (antes de tiempo)

`NODOCLIENTE1` envía un `MsgPeticionPrimario` al GV que debe ser respondido con un `HOSTINDEFINIDO` ya que en la primera vista no se cuenta con ningún nodo primario ni copia.

3.2. Hay un primer primario correcto

`NODOCLIENTE1` envía un `MsgLatido` con argumento 0 al GV que debe ser respondido con una primera vista tentativa en la que él mismo es el primario.

3.3. Hay un nodo copia

`NODOCLIENTE1` envía un `MsgLatido` con argumento -1 (situación excepcional explicada en sección 2.1) al GV para señalar que sigue vivo. Posteriormente, `NODOCLIENTE2` envía un `MsgLatido` con argumento 0 que debe ser respondido con una segunda vista tentativa en la que él mismo es la copia.

3.4. Copia toma relevo si primario falla

`NODOCLIENTE1` envía un `MsgLatido` con argumento 2 al GV para confirmar la vista (estado consistente). Posteriormente, el GV detecta la caída del primario (ha dejado de recibir latidos) y promociona la copia a primario en una tercera vista. Si el primario no hubiese confirmado la vista antes de caer, al estar en un estado inconsistente, se habría producido un fallo crítico.

3.5. Servidor rearrancado se convierte en copia

`NODOCLIENTE1` envía un `MsgLatido` con argumento 0 para comunicar que ha rearrancado. El GV lo introduce en la lista de servidores en espera y, al comprobar que no existe nodo copia, lo convierte en copia en una cuarta vista.

3.6. Servidor en espera se convierte en copia si primario falla

`NODOCLIENTE2` envía un `MsgLatido` con argumento 4 al GV para confirmar la vista (estado consistente). Posteriormente, `NODOCLIENTE3` envía un `MsgLatido` con argumento 0 al GV y es añadido a la lista de servidores en espera.

GV detecta la caída del nodo primario al recibir únicamente latidos de `NODOCLIENTE1` y `NODOCLIENTE3`, y crea una quinta vista tentativa promocionando la copia a primario, y el nodo en espera a copia.

3.7. Primario rearrancado es tratado como caído y convertido en nodo en espera

`NODOCLIENTE1` envía un `MsgLatido` con argumento 5 al GV para confirmar la vista (estado consistente). Y `NODOCLIENTE2` envía un `MsgLatido` con argumento 0 al GV y es añadido a la lista de servidores en espera. Posteriormente, el GV detecta la caída del primario (latido con argumento 0) y crea una sexta vista promocionando la copia a primario, y el nodo espera a copia.

3.8. Servidor de vistas espera a que primario confirme vista, pero este no lo hace

El GV no confirmará la vista tentativa hasta que el primario se lo comunique. Por lo tanto, la vista válida seguirá siendo la quinta hasta que el primario no envíe un latido con argumento 6.

3.9. Si anteriores servidores caen, un nuevo servidor sin inicializar no puede convertirse en primario

El GV detecta la caída simultánea del nodo primario y copia y genera un error fatal. Por lo tanto, ningún otro nodo le podrá enviar `MsgLatido` convirtiéndose en primario.

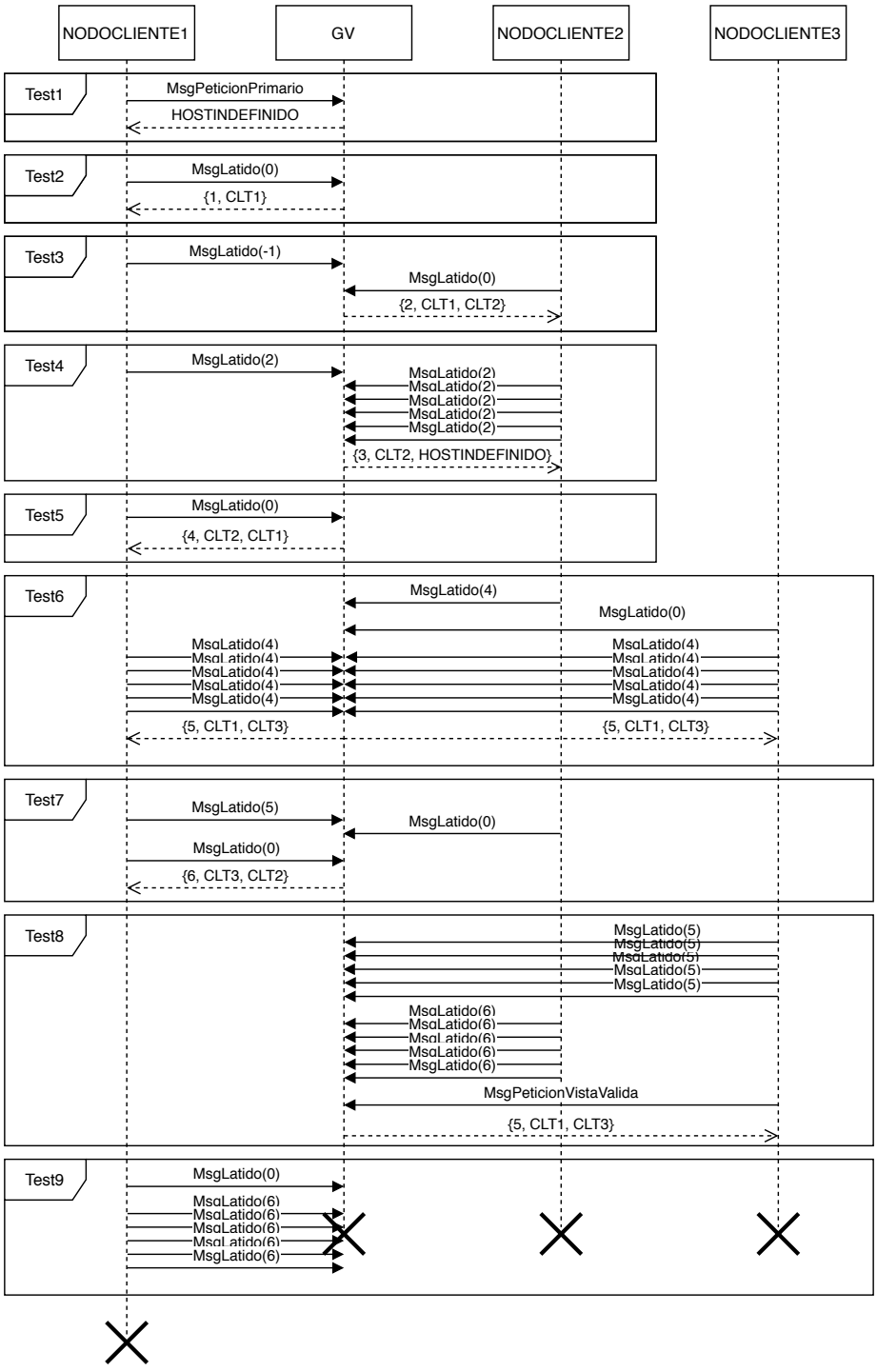

Figura 1: Diagrama de secuencia

4. Conclusiones

Tras la realización de esta práctica se ha constatado el aumento de dificultad a la hora de crear un sistema distribuido resistente a fallos cuando se requiere de estado, debido a la necesidad de asegurar consistencia en el estado.

Se ha visto igualmente la importancia de realizar test al código que se genera, permitiendo la modificación del código existente asegurando que no se modifica el comportamiento deseado. Aunque esto genere otro problema, la dificultad en la creación de test que verifiquen correctamente la funcionalidad requerida y la falsa seguridad que esto conllevaría.