Sistemas Distribuidos
Práctica 5: Servicio de Primary/Backup mediante clave/valor
Memoria

Javier Beltrán Jorba, 532581
Jorge Cáncer Gil, 646122

Índice

Introducción

El objetivo de esta práctica es desarrollar un sistema de servidores clave/valor al que los clientes puedan acceder para leer o escribir datos. Para garantizar la tolerancia a fallos, se va a utilizar replicación primario/*backup* para solucionar problemas originados por caídas de red, servidores que fallan, etc.

Sin embargo, no se va a tener en cuenta la necesidad de replicar el gestor de vistas, por lo que el sistema no será totalmente tolerante a fallos.

Objetivos del sistema

El servidor clave/valor almacena una serie de pares (clave,valor) en memoria, y proporciona a los clientes una interfaz con las siguientes operaciones para utilizar el servicio:

- Leer: el cliente indica la clave cuyo valor quiere leer, y recibe dicho valor o un mensaje de error.
- Escribir: el cliente indica la clave que quiere modificar junto con el nuevo valor que le quiere asignar, y recibe la confirmación de que su escritura se ha producido (o un mensaje de error).
- Escribir_hash: el cliente indica la clave que quiere modificar junto con el nuevo valor que le quiere asignar, y escribe el resultado de aplicar la concatenación del valor antiguo y el nuevo. Además, el sistema le devuelve el valor antiguo (o un mensaje de error).

Para garantizar la tolerancia a fallos, se ha replicado este servidor (que llamaremos primario), por lo que existe una copia de sus datos en un servidor de *backup*. Además pueden existir servidores en espera, que entrarán en funcionamiento cuando algún servidor en uso caiga.

La existencia de múltiples servidores con distintas funcionalidades la gestiona el servidor de vistas, que mantiene una secuencia de vistas con un servidor primario y uno de *backup*. Los clientes acceden al gestor de vistas para conocer quién es el primario al que deben realizar las peticiones. Además, éste gestor garantiza la tolerancia a fallos creando una nueva vista cuando detecte que un servidor ha caído.

Diseño del sistema

Si se descompone el escenario en componentes, aparecen algunos elementos principales:

- Los clientes, que se conectarán desde sus máquinas a los servidores.
- El servidor de vistas, al que los clientes se conectarán para conocer a qué servidor deben dirigirse.
- Los servidores clave/valor, conectados con el servidor de vistas y que deberán aceptar las peticiones de los clientes si toman el rol de primario. Además, necesitan conocer al servidor de *backup* para replicar las operaciones realizadas.

Sin embargo, al ser necesario incluir un sistema de latidos que informe periódicamente al gestor de vistas de que cada servidor está vivo, se ha decidido añadir al gestor de vistas una serie de procesos de *timeout*. Cada uno de ellos controla que el servidor de vistas recibe los latidos de un servidor clave/valor a tiempo. En caso de que un servidor deje de enviar 5 latidos, este proceso informará al gestor de que está caído.

Del mismo modo, en el lado del servidor clave/valor es necesaria una entidad que esté enviando periódicamente los latidos. Por ello, cada servidor lleva asociado un emisor de latidos.

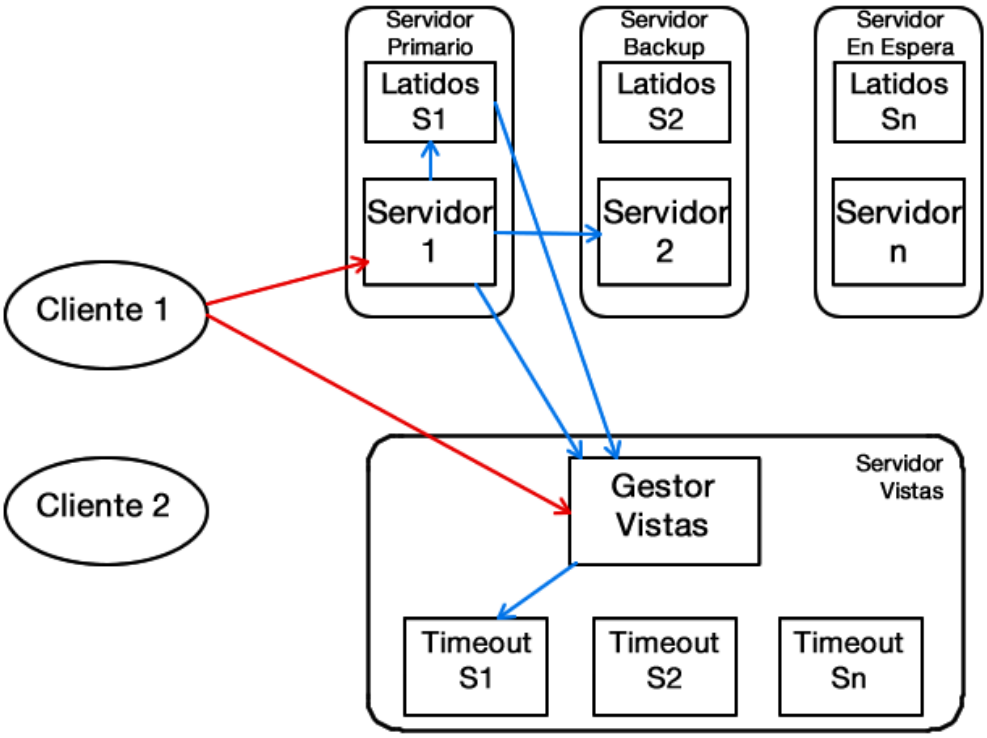

Figura 1: Esquema de componentes del sistema. Las flechas azules reflejan los canales de comunicación del servidor 1, y las rojas indican las interacciones del cliente 1.

El sistema resultante tras incluir estos elementos se refleja en la figura 1. Se trata de una representación simplificada (solo incluye las interacciones del servidor primario) que permite comprender los diferentes elementos de este diseño y su funcionalidad.

Protocolos de interacción

Una vez conocidos los elementos implicados, es necesario explicar cómo se comunican entre ellos. Para esto se han definido una serie de protocolos, que se van a explicar a continuación mediante diagramas de secuencia y que representan las principales situaciones del sistema.

Sistema de latidos

Cuando un servidor clave/valor se registra en el sistema, debe comunicarse mediante el latido 0 con el gestor de vistas, que decidirá qué rol asignarle y se lo comunicará con la vista válida.

Desde ese momento, el servidor clave/valor debe enviarle cada 100ms un latido al gestor para demostrarle que está vivo. Del envío de estos mensajes se encarga un proceso autónomo, que solo se detendrá si se cae el servidor. Del mismo modo, el gestor creará un proceso que temporice la llegada de latidos, y que le avise cuando detecte que el servidor se ha caído.

El gestor de vistas tolera hasta 5 latidos consecutivos perdidos antes de declarar que un servidor ha dejado de funcionar. Cuando esto suceda, creará una nueva vista y la comunicará a los servidores vivos en su siguiente latido. La figura 2 refleja, mediante un diagrama de secuencia, el envío periódico de latidos que termina en una situación de caída.

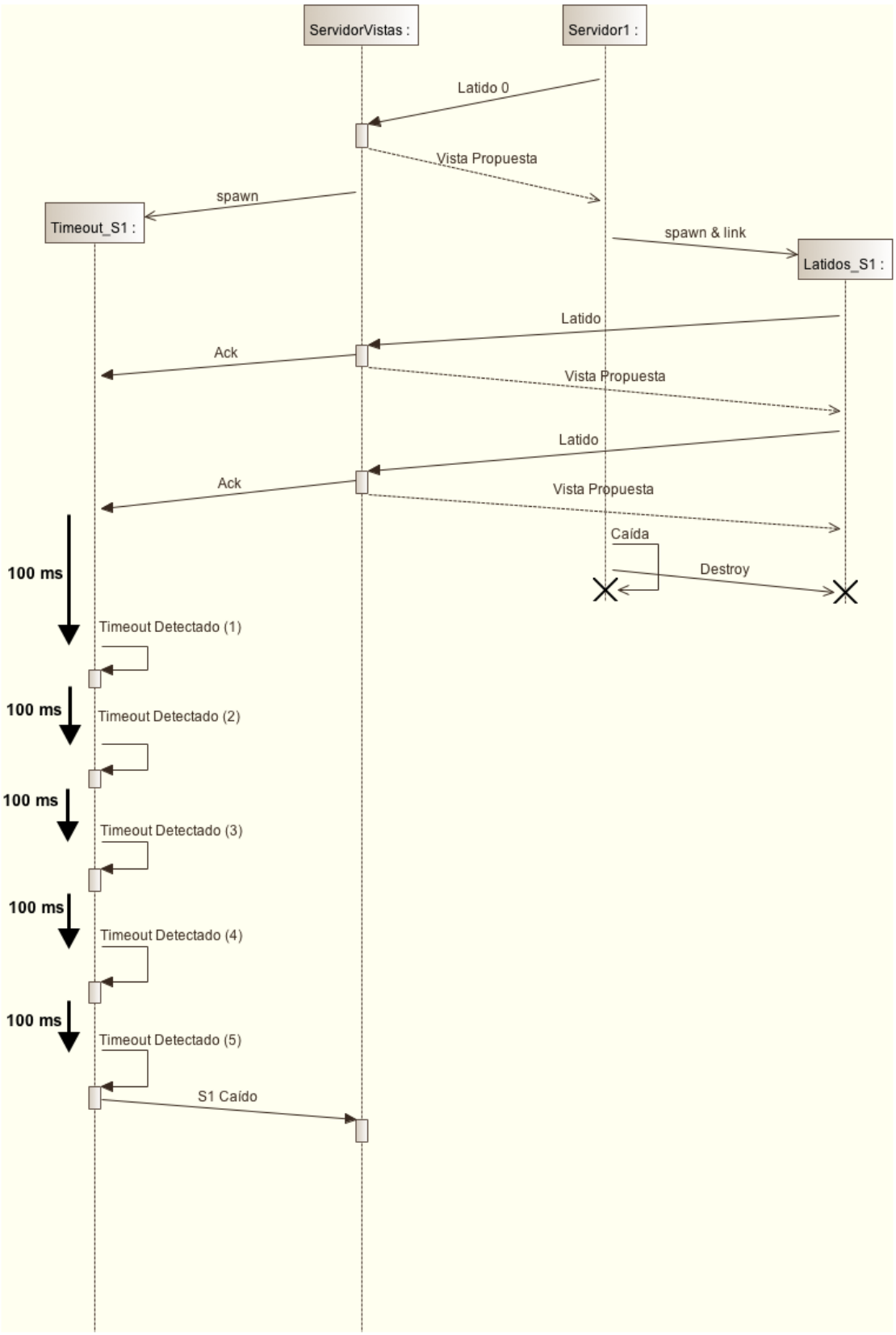

Figura 2: interacción entre el servidor clave/valor y el gestor de vistas mediante un sistema de latidos que demuestra si está vivo o muerto

Leer y escribir

El sistema solo aceptará peticiones de los clientes si existe un servidor de *backup* activo, para garantizar la consistencia de los datos. De este modo, una petición de escritura de un dato es gestionada por el servidor primario propagándola al servidor *backup*, que actualizará su diccionario.

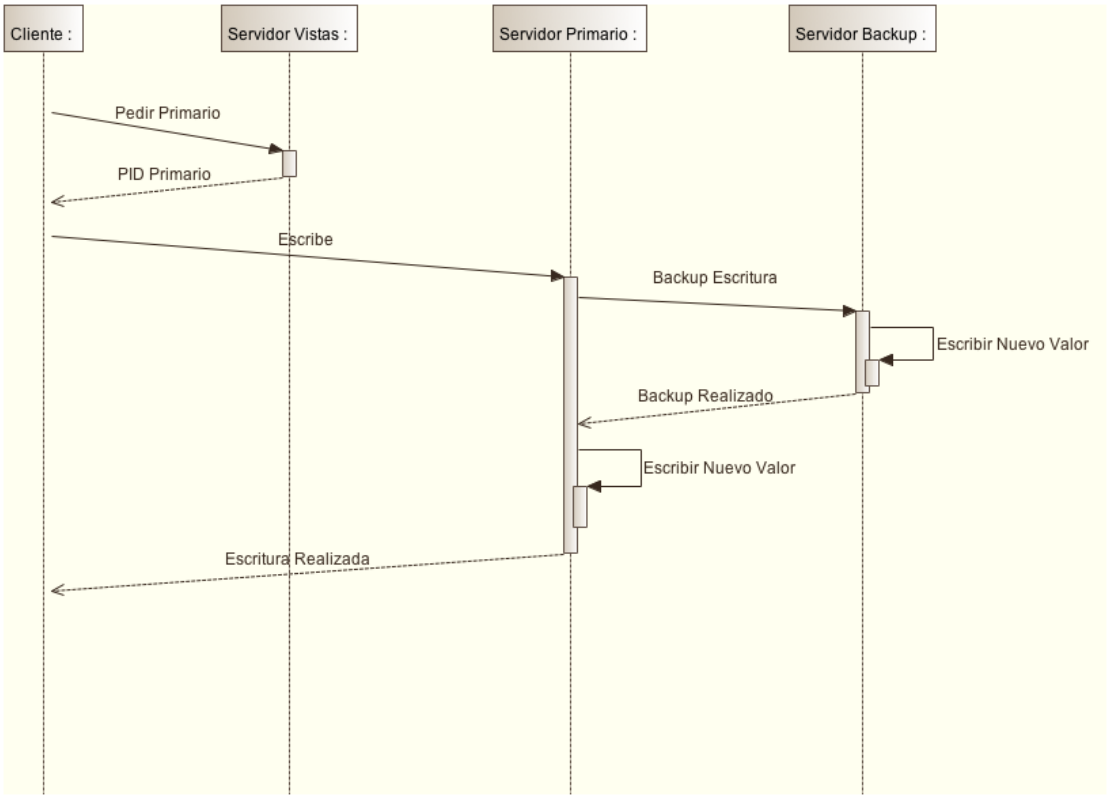

Figura 3: Petición de escritura de un dato y propagación al backup

A continuación el servidor primario actualizará el suyo, y solo entonces (cuando se garantiza que todos los servidores han alcanzado el mismo estado) avisa al cliente de que la escritura se ha realizado correctamente. La figura 3 muestra este proceso.

En el caso de las lecturas, el mecanismo es idéntico, y este proceso se explica en la figura 4.

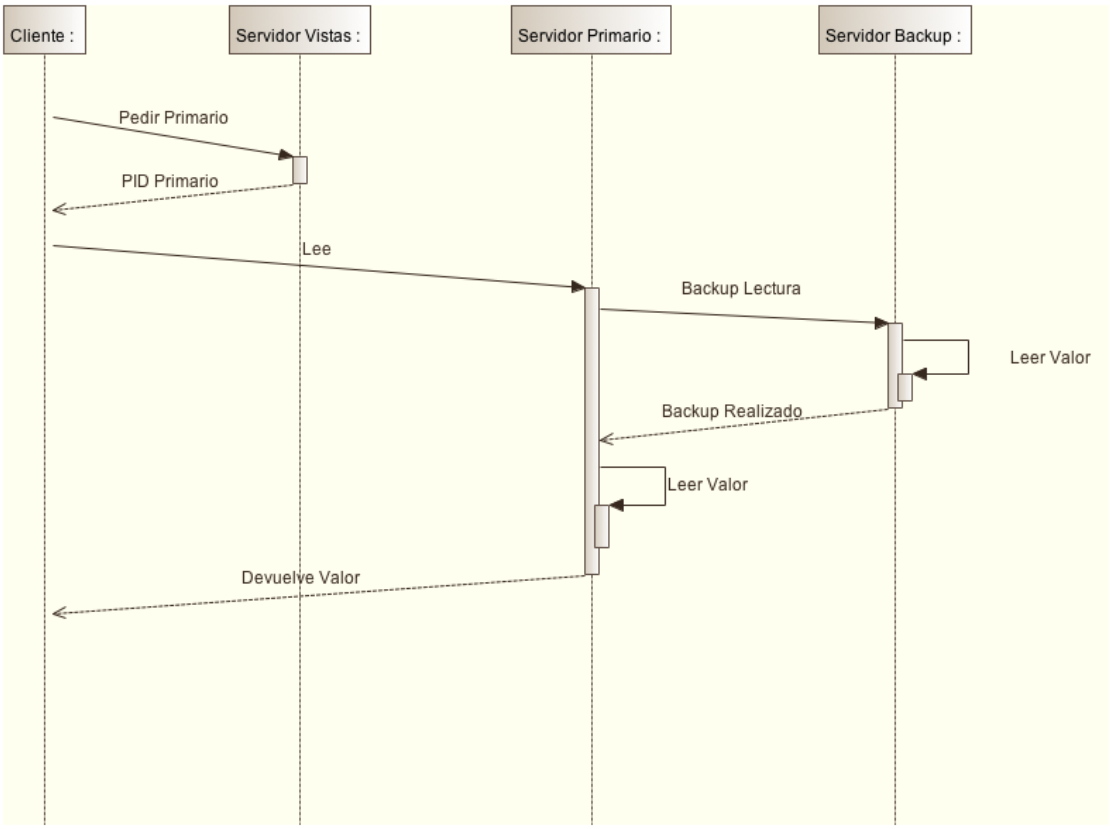

Figura 4: Petición de lectura de un dato y propagación al backup

Tolerancia a fallos

Para garantizar que el sistema siga funcionando correctamente en una situación de error (y que lo haga de forma transparente al usuario) se han diseñado unos protocolos de actuación en caso de fallo de alguno de los servidores clave/valor. Como se ha indicado anteriormente, no se contempla el caso de fallo del gestor de vistas.

La primera de las situaciones anómalas que se han tenido en cuenta es la **falta de un servidor de *backup*** al que el servidor primario pueda propagar las operaciones realizadas. Gracias a la transferencia de la vista válida, es consciente de esta situación y puede responder a las peticiones de los clientes con un mensaje de error.

La siguiente situación de error detectada es la **caída del servidor primario**. En este caso el cliente nunca recibirá una respuesta a su petición. Por ello, debe estar preparado con un *timeout* para no quedarse bloqueado permanentemente. En caso de que salte el *timeout*, deberá volver a preguntar por el servidor primario al gestor de vistas, pues éste habrá cambiado.

La figura 5 representa la situación de falta de *backup*, seguida de una escritura correcta, y posteriormente una situación de caída del servidor primario.

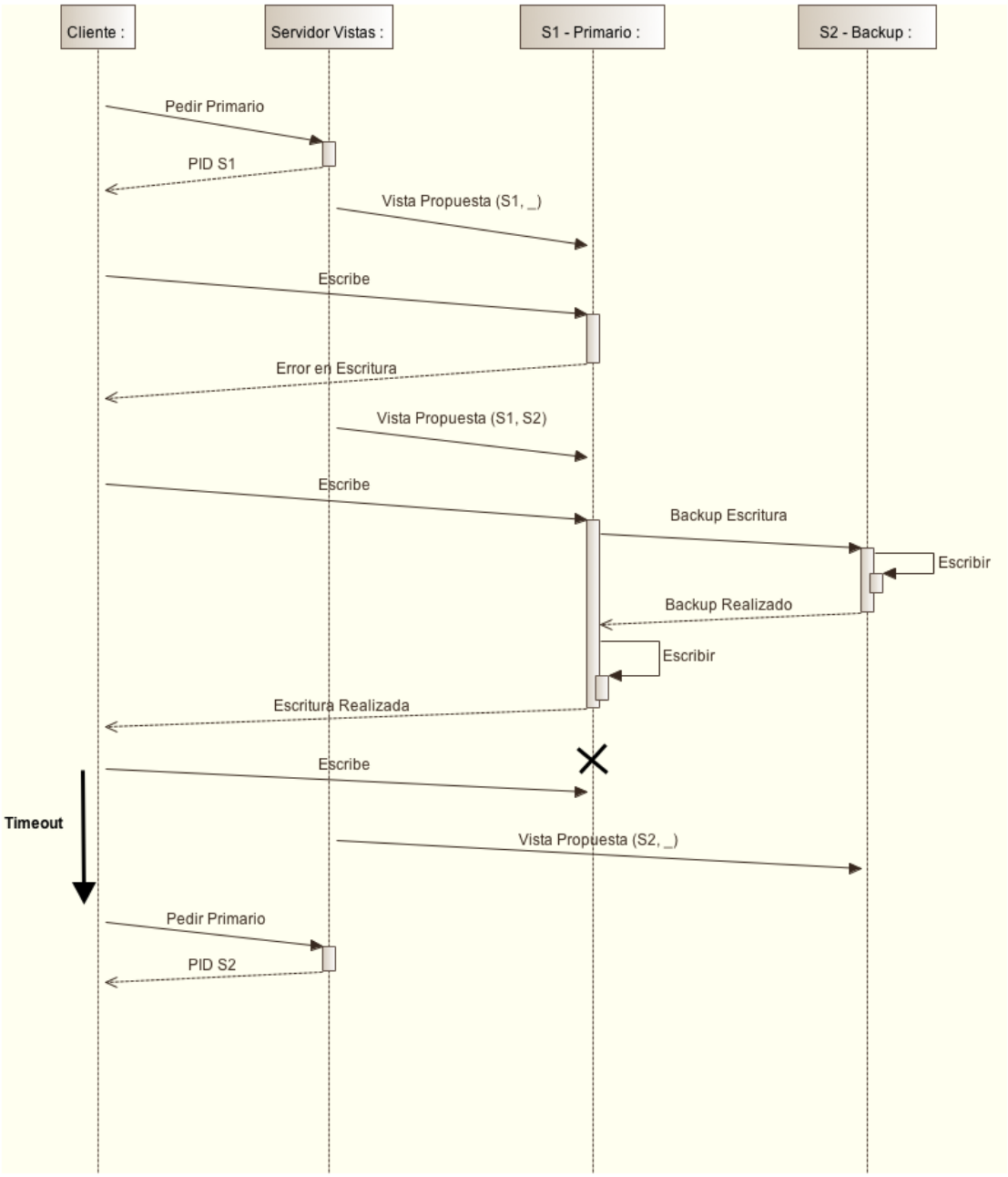

Figura 5: Error de escritura por no existir un servidor de backup, y error por caída del servidor primario

Se ha tratado el caso de caída del servidor primario, pero el servidor de *backup* también puede caer. En este caso puede haber dos situaciones. La primera consiste en que el servidor primario **recibe un vista propuesta en la que primario y *backup* funcionan correctamente**, e inmediatamente recibe una petición de un cliente, pero simultáneamente el **servidor de *backup* falla.**

El servidor primario no se enterará de la caída hasta que reciba la siguiente vista, pero antes deberá gestionar la petición del cliente. Será entonces, al propagar la operación de lectura o escritura al *backup*, cuando no reciba respuesta. Por ello se ha de programar un *timeout* en el servidor primario de modo que no se quede bloqueado, sino que avise al cliente de que ha habido un error.

Otra situación posible es que el servidor primario detecte que ha habido un cambio de vista debido a la **caída del servidor de *backup*, por lo que debe replicar su diccionario** para mantener la consistencia. Hasta que el nuevo *backup* no le informe de que se ha completado la replicación, el servidor primario no puede aceptar peticiones. Por ello, toda petición que llegue durante ese intervalo de tiempo debe ser cancelada directamente.

La figura 6 refleja ambos casos de fallo, seguidos de una petición de lectura correcta. En el diagrama se aprecia la necesidad de incluir un *timeout* en el servidor primario para garantizar que no se queda bloqueado esperando a un servidor *backup* caído.

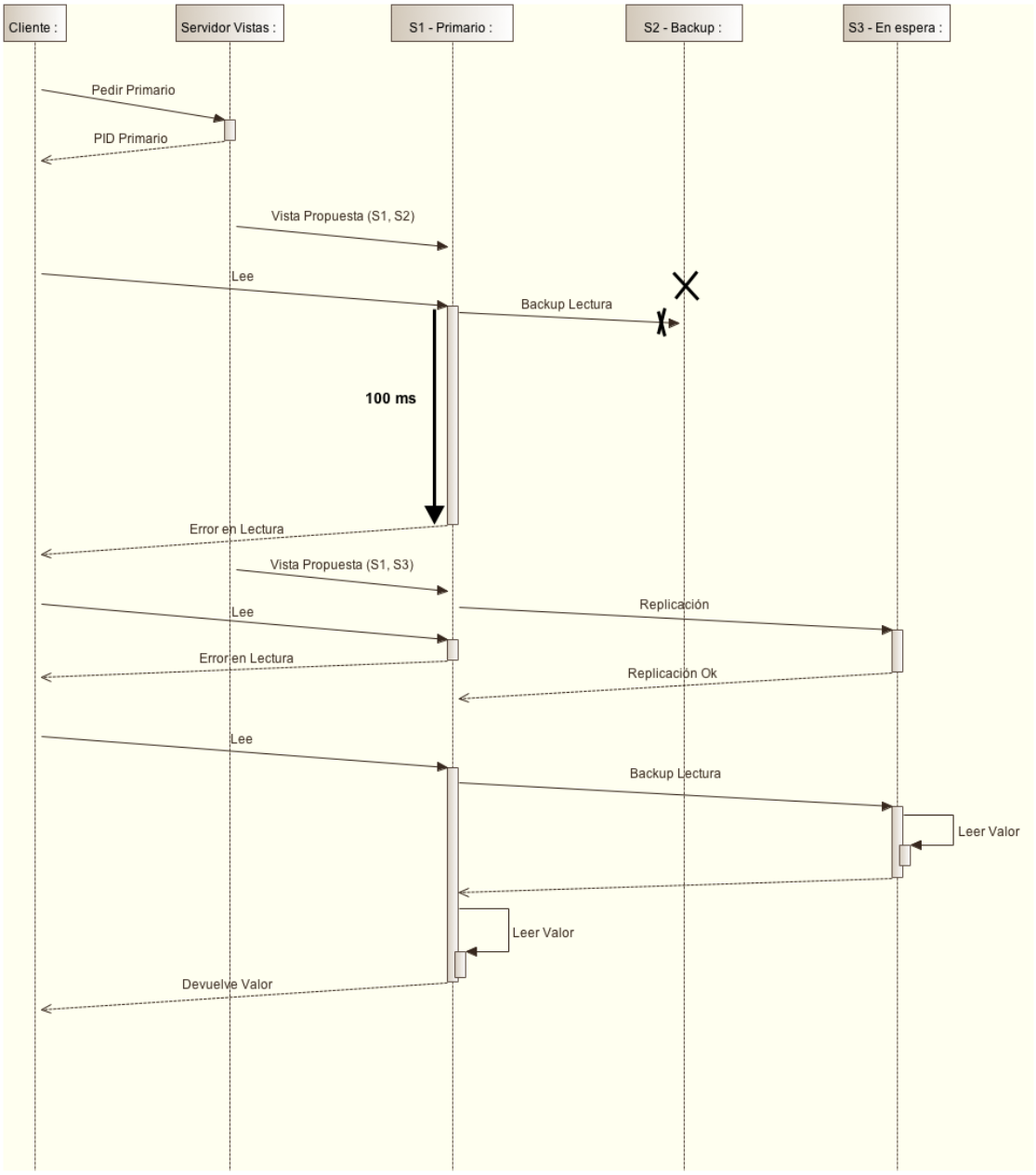

Figura 6: Situación de error en la que el *backup* se cae y el primario aún no lo sabe, seguida de situación de replicación de los datos por la entrada de un nuevo *backup*.

Validación

Servicio de vistas

En primer lugar, se han repetido las comprobaciones del primer apartado de la práctica para asegurar la correcta integración de todas las partes. Los resultados han sido los esperados, no habiendo ningún error. Se adjunta la traza completa a continuación.

```
Arrancando servidor de vistas
Lanzando latedor
Enviado latido inicial... -> 1
Latido inicial de 1
 --> Pasa a ser: Primario
Vista propuesta actual: 1 1 0
<0.343.0>
Recibida primera respuesta -> 1
Iniciando replicacion
No hay backup para replicarse
Lanzando latedor
Enviado latido inicial... -> 2
Latido inicial de 2
 --> Pasa a ser: Backup
Vista propuesta actual: 2 1 2
<0.343.0>
Recibida primera respuesta -> 2
Iniciando replicacion
BU recibido
Replicación completada
Sin latido de 1, intentando Timeout 4
Sin latido de 1, intentando Timeout 3
Sin latido de 1, intentando Timeout 2
Sin latido de 1, intentando Timeout 1
Sin latido de 1, intentando Timeout 0
Caido servidor 1 --> Rol: Primario
2 pasa a ser primario
Buscando backup del proceso...
<0.343.0>
Hay 1 servidores activos.
No hay servidores en espera.
Vista propuesta actual: 3 2 0
<0.343.0>
Iniciando replicacion
No hay backup para replicarse
Lanzando latedor
Enviado latido inicial... -> 1
Latido inicial de 1
 --> Pasa a ser: Backup
El servidor 1 ha rearrancado
Recibida primera respuesta -> 1
Vista propuesta actual: 4 2 1
<0.343.0>
Iniciando replicacion
BU recibido
Replicación completada
Lanzando latedor
Enviado latido inicial... -> 3
Latido inicial de 3
 --> Pasa a ser: Espera
Vista propuesta actual: 4 2 1
<0.343.0>
Replicación completada
Recibida primera respuesta -> 3
Sin latido de 2, intentando Timeout 4
Sin latido de 2, intentando Timeout 3
Sin latido de 2, intentando Timeout 2
Sin latido de 2, intentando Timeout 1
Sin latido de 2, intentando Timeout 0
Caido servidor 2 --> Rol: Primario
1 pasa a ser primario
Buscando backup del proceso...
<0.343.0>
Hay 2 servidores activos.
3 pasa a ser backup.
Vista propuesta actual: 5 1 3
<0.343.0>
Iniciando replicacion
BU recibido
Replicación completada
Lanzando latedor
Enviado latido inicial... -> 2
true
Latido inicial de 2
 --> Pasa a ser: Espera
El servidor 2 ha rearrancado
Recibida primera respuesta -> 2
Vista propuesta actual: 5 1 3
<0.343.0>
Sin latido de 3, intentando Timeout 4
Sin latido de 3, intentando Timeout 3
Sin latido de 3, intentando Timeout 2
Sin latido de 3, intentando Timeout 1
Sin latido de 3, intentando Timeout 0
Caido servidor 3 --> Rol: Backup
Buscando backup del proceso...
<0.343.0>
Hay 2 servidores activos.
2 pasa a ser backup.
Vista propuesta actual: 6 1 2
<0.343.0>
Iniciando replicacion
BU recibido
Replicación completada
```

Peticiones sin *backup*

Se ha comprobado que las peticiones de los clientes fallan cuando ningún servidor ocupa el puesto de *backup*. La traza que lo demuestra es la siguiente.

```
Arrancando servidor de vistas
Lanzando latedor
Enviado latido inicial... -> 1
Latido inicial de 1
 --> Pasa a ser: Primario
Vista propuesta actual: 1 1 0 <0.60.0>
Recibida primera respuesta -> 1
Iniciando replicacion
No hay backup para replicarse
Conociendo primario
Recibida peticion de primario. Es: 1
Primario <0.61.0>
Enviando 1 mensaje1 <0.60.0>
Error en la escritura del valor : mensaje1<0.32.0>
```

Escritura tras fallo de *backup*

Para poder comprobar cómo reacciona el servicio a una petición tras un fallo de *backup* se ha introducido un retardo de 1 segundo en medio del proceso de replicación al nuevo *backup*. De este modo se puede comprobar qué sucede si una petición llega en pleno proceso de replicación.

Es lo que sucede con el primer mensaje de la traza: “ahora no”. Llega antes de completarse la replicación así que no puede escribirse. Sin embargo, el mensaje “ahora sí”, que llega después, sí que se escribe.

```
Arrancando servidor de vistas
Lanzando latedor
Enviado latido inicial... -> 1
Latido inicial de 1
 --> Pasa a ser: Primario
Vista propuesta actual: 1 1 0 <0.91.0>
Recibida primera respuesta -> 1
Iniciando replicacion
No hay backup para replicarse
Lanzando latedor
Lanzando latedor
Enviado latido inicial... -> 2
Enviado latido inicial... -> 3
Latido inicial de 2
 --> Pasa a ser: Backup
Vista propuesta actual: 2 1 2 <0.91.0>
Recibida primera respuesta -> 2
Latido inicial de 3
 --> Pasa a ser: Espera
Vista propuesta actual: 2 1 2 <0.91.0>
Recibida primera respuesta -> 3
Iniciando replicacion
BU recibido
Replicación completada
Sin latido de 2, intentando Timeout 4
Sin latido de 2, intentando Timeout 3
Sin latido de 2, intentando Timeout 2
Sin latido de 2, intentando Timeout 1
Sin latido de 2, intentando Timeout 0
Caido servidor 2 --> Rol: Backup
```

```
Buscando backup del proceso... <0.91.0>
Hay 2 servidores activos.
3 pasa a ser backup.
Vista propuesta actual: 3 1 3 <0.91.0>
Iniciando replicacion
BU recibido
Conociendo primario
Iniciando replicacion
BU recibido
Conociendo primario
Recibida peticion de primario.
Primario <0.92.0>
Enviando 1 ahora no <0.91.0>
Error en la escritura del valor : ahora no
<0.32.0>
Replicación completada
Conociendo primario
Recibida peticion de primario.
Primario <0.92.0>
Enviando 1 ahora si <0.91.0>
BU recibido
Procesando respaldo 2
Recibida confirmacion del BackUp
Escrito correctamente el valor : ahora si
<0.32.0>
```

Escritura tras fallo de primario

Este caso es muy parecido al anterior. El servidor de *backup* pasa a ser primario y realiza la replicación a un nuevo servidor de *backup*. Se han realizado las mismas dos peticiones: una en pleno proceso de replicación y otra después de éste, y los resultados son los mismos que antes.

```
Arrancando servidor de vistas
Lanzando latedor
Enviado latido inicial... -> 1
Latido inicial de 1
 --> Pasa a ser: Primario
Vista propuesta actual: 1 1 0 <0.102.0>
Recibida primera respuesta -> 1
Iniciando replicacion
No hay backup para replicarse
Lanzando latedor
Lanzando latedor
Enviado latido inicial... -> 2
Enviado latido inicial... -> 3
Latido inicial de 2
 --> Pasa a ser: Backup
Vista propuesta actual: 2 1 2 <0.102.0>
Recibida primera respuesta -> 2
Latido inicial de 3
 --> Pasa a ser: Espera
Vista propuesta actual: 2 1 2 <0.102.0>
Recibida primera respuesta -> 3
Iniciando replicacion
BU recibido
Replicación completada
Sin latido de 1, intentando Timeout 4
Sin latido de 1, intentando Timeout 3
Sin latido de 1, intentando Timeout 2
Sin latido de 1, intentando Timeout 1
Sin latido de 1, intentando Timeout 0
Caido servidor 1 --> Rol: Primario
2 pasa a ser primario
Buscando backup del proceso... <0.102.0>
Hay 2 servidores activos.
3 pasa a ser backup.
Vista propuesta actual: 3 2 3 <0.102.0>
Iniciando replicacion
BU recibido
Conociendo primario
Recibida peticion de primario.
Primario <0.106.0>
Enviando 1 ahora no <0.102.0>
Error en la escritura del valor : ahora no
<0.32.0>
Replicación completada
Conociendo primario
Recibida peticion de primario.
Primario <0.106.0>
Enviando 1 ahora si <0.102.0>
BU recibido
Procesando respaldo 2
Recibida confirmacion del BackUp
Escrito correctamente el valor : ahora si
<0.32.0>
```

Escrituras concurrentes

Si dos clientes envían mensajes de escritura de manera concurrente sobre la misma clave, uno de los dos mensajes llegará antes. Ese será el primero en ser ejecutado, y no se atenderá a la otra escritura hasta que no se haya replicado y confirmado la anterior.

De este modo, se garantiza mantener la consistencia de los datos. La primera escritura se perderá porque instantáneamente será sobrescrita, pero es algo inherente a un sistema en el que cualquier cliente puede actualizar un dato en cualquier momento.

```
Arrancando servidor de vistas
Lanzando latedor
Enviado latido inicial... -> 1
Latido inicial de 1
 --> Pasa a ser: Primario
Vista propuesta actual: 1 1 0 <0.144.0>
Recibida primera respuesta -> 1
Iniciando replicacion
No hay backup para replicarse
Lanzando latedor
Enviado latido inicial... -> 2
Latido inicial de 2
 --> Pasa a ser: Backup
Vista propuesta actual: 2 1 2 <0.144.0>
Recibida primera respuesta -> 2
Iniciando replicacion
BU recibido
Replicación completada
Conociendo primario
Recibida peticion de primario. Es: 1
Primario <0.145.0>
Enviando 1 soy 1 <0.144.0>
BU recibido
Procesando respaldo 2
Recibida confirmacion del BackUp
Escrito correctamente el valor : soy 1<0.32.0>
Conociendo primario
Recibida peticion de primario. Es: 1
Primario <0.145.0>
Enviando 1 soy 2 <0.144.0>
BU recibido
Procesando respaldo 2
Recibida confirmacion del BackUp
Escrito correctamente el valor : soy 2<0.32.0>
Recibida peticion de primario. Es: 1
Conociendo vista valida
Vista a enviar 2 <0.145.0> <0.148.0>
Iniciando respaldo de lectura
Recibida confirmacion del BackUp
Leido valor : soy 2 <0.32.0>
```

El resultado de una lectura posterior a ambas escrituras siempre será el segundo valor escrito. Sin embargo, el orden de las escrituras dependerá de las condiciones de carrera en cada caso.

Partición de red con 2 primarios vivos

Problemas de red pueden provocar que dos servidores se crean primarios pero que solo uno de ellos lo sea realmente (porque así ha sido nombrado por el gestor de vistas). El problema asociado a estos casos es que este servidor confundido acepte peticiones de los clientes y la consistencia de estas operaciones no se garantice.

Puesto que el gestor de vistas resolverá esta situación de fallo de red tratando al servidor primario o al de *backup* como caídos, la vista cambiará. Se puede afirmar que en ambos casos el servidor de *backup* cambiará su posición (ya sea para ser primario o para estar en espera, dependiendo de a quién de los dos servidores crea como caído el gestor de vistas).

Así, se puede aprovechar esta situación y el hecho de que todas las operaciones se replican en nuestro diseño para garantizar que las peticiones a un servidor que se cree primario (pero no lo es) no se ejecuten. Basta con asegurar que, si un servidor recibe la replicación de una operación, éste solo la acepte si toma el rol de *backup*.

De este modo, cuando el servidor propague una petición a su *backup*, éste ya no lo será, pues la vista habrá cambiado. En consecuencia, responderá con un error a la replicación, y el servidor que se creía primario le devolverá error al cliente.

Dada la dificultad para alcanzar esta situación en un entorno de pruebas, no se ha podido dar una traza de ejecución. Sin embargo el código está preparado para este tipo de casos.

Validación en múltiples nodos

Con el objetivo de hacer una validación lo más cercana posible a un caso real, se ha ejecutado el sistema en múltiples nodos de Erlang. De ese modo se ha podido simular qué sucede si una de las máquinas se cae repentinamente, por ejemplo. Los resultados han sido los esperados, quedando demostrado que el sistema funciona igual en un entorno real que en una sola máquina.

Conclusiones

Se ha diseñado un sistema distribuido en el que los clientes pueden enviar peticiones de lectura y escritura de datos almacenados en los servidores. Éstos están replicados siguiendo la técnica de primario/*backup*, para garantizar que el servicio es tolerante a fallos.

En resumen, se ha conseguido un sistema robusto y transparente para el usuario ante los fallos que pueden acontecer. Sin embargo, no se han tratado todos los errores: el gestor de vistas no está replicado, por lo que sigue existiendo un punto de fallo. Soluciones más completas como Raft sí que ofrecen tolerancia a fallos completa.