Sistemas distribuidos

Práctica 5.

1º parte: Servicio de gestión de vistas

Daniel Martínez Martínez, 538798

Samuel Ruiz de Gopegui Muñoz, 685127

16 de diciembre de 2016

Diseño del sistema

Implementación de un sistema con tolerancia a fallos mediante gestor de vistas, usando replicación Primario/Copia en RAM. Se ha diseñado el siguiente autómata para el sistema gestor de vistas [Fig.1]. En este diagrama existe un caso bloqueante y es una vez inicializado, cuando el sistema en un momento dado no tiene copia y el primario cae, el sistema se bloquea. Se ha planteado que en ese estado el sistema devuelva que no hay primario ni copia y el sistema se bloquea (datos :undefined).

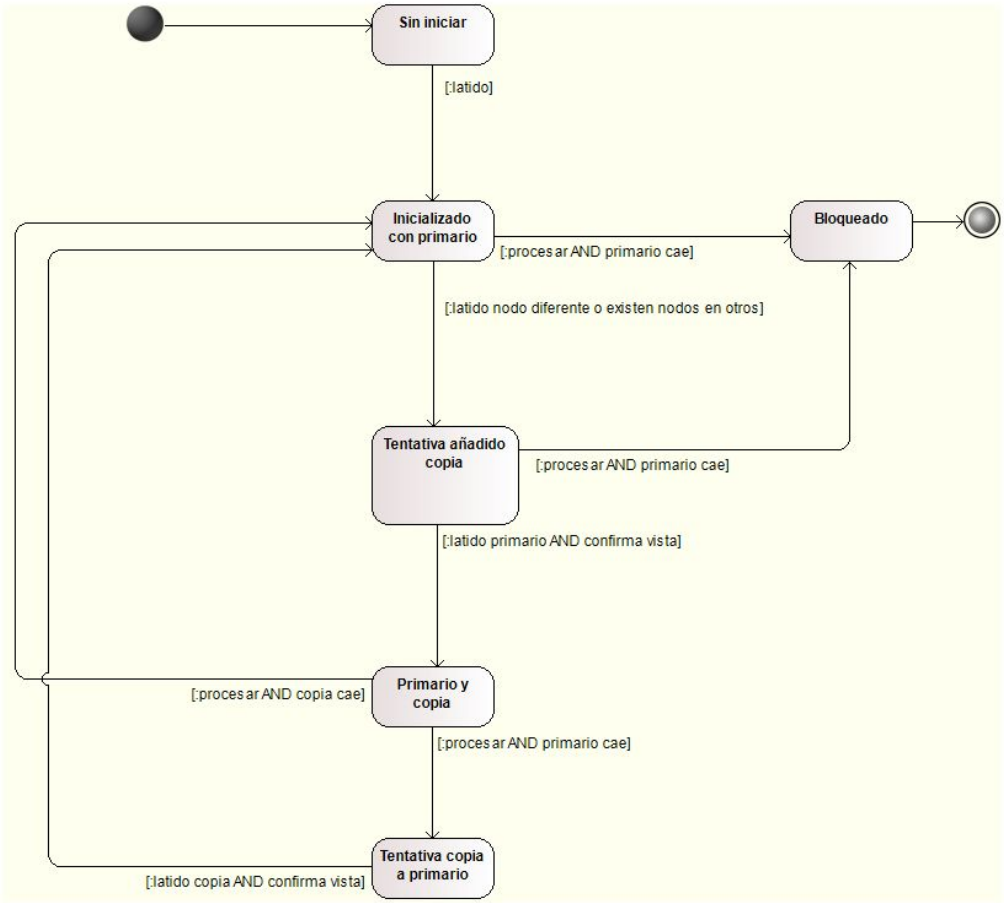


Fig. 1: Máquina de estados del sistema gestor de vistas.

El gestor de vistas gestiona una secuencia de “vistas” numeradas según -nº de vista, identificador primario, identificador copia- para cada vista.
Al iniciar el servicio, el nodo primario puede ser cualquier nodo, haciendo una copia del mismo en cualquier otro nodo, o puede no existir en momentos específicos.

El gestor de vistas controla cada nodo y su copia, de manera que si el nodo primario falla, y no hay un nodo en espera, su copia es promocionada, si existe un nodo en espera, este es promocionado.

Se crean nuevas vistas si el nodo primario no ha enviado un latido durante un intervalo y este tiene una copia, o en el caso de que la copia cae. Asimismo no se activa la vista tentativa hasta que el respectivo responsable envíe el ping con el número de vista correcto a dicha tentativa.

Implementación del sistema

Se ha definido una estructura en el servidor de vistas que almacena el estado de la vista con los siguientes campos:

- vista_v
- vista_t
- vidas → vidas de todos los nodos
- otros → otros servidores
- reloj → almacena el latido más reciente
- init → estado iniciado
- confirmar
- valido

Como el sistema tiene que responder ante eventos de clientes en el momento en el que se procesan los diferentes mensajes. Se ha planteado que parte de la lógica esté en la función bucle_recepcion, que el cual al recibir :latidos aumenta las vidas de los nodos. Y si se trata de un nodo primario o responsable de confirmar una nueva tentativa, se cambiará el estado del servidor.

La otra parte de la lógica de negocio reside en la función procesa, la cual realiza las funciones de mantenimiento para garantizar que el servidor detecta los fallos de los nodos involucrados. Las funciones que realiza son restar vidas a los nodos, comprobar que mediante estas vidas el primario o copia aun siguen vivos, de proponer nuevas tentativas ante estas situaciones.

Por otro lado mensajes como :obten_vista, envía al nodo con un pid dado con el estado de la vista en curso (si es válida o no). A este se le responde con el mensaje :vista_valida, y si recibe :procesa_situacion_servidores, llama a la función procesas_situacion_servidores enviandole el estado del nodo.

Validación

Como parte del trabajo se ha proporcionado un diagrama de secuencia en el que se explica la evolución de las diferentes pruebas realizadas para los primeros casos [Fig.2]. Se han completado las pruebas restantes y ejecución el código.

En el diseño de las pruebas se han tenido algunas consideraciones para el correcto funcionamiento.

Los distintos nodos, al enviar un latido, si envian número de vista 0, están indicando que no siguen el orden o se han caído. Esto provoca que el servidor gestor de vistas considere si eran primarios o copia que han caído. Con lo que se ha establecido que para indicar que no se está confirmando nada y solo que están vivos, que se envíe un número -1 o negativo.

Durante el desarrollo de la práctica se ha percatado que se envía un booleano desde el servidor ante los diferentes mensajes. Por el código de las pruebas se ha supuesto que será usado para confirmar si la vista obtenida es correcta o en caso contrario para indicar que el servidor no está en funcionamiento. En la práctica se envia cierto o falso cuando el servidor está en modo bloqueado (que es lo mismo que el primario ha caído cuando no tenía una copia confirmada).

Pruebas realizadas
1. Copia toma el relevo si primario falla.
2. Servidor rearrancado se convierte en copia.
3. 3er servidor en espera se convierte en copia si primario falla.
4. Primario rearrancado es tratado como caido.
5. Servidor de vistas espera a que primario confirme vista, pero este no lo hace.
6. Si anteriores servidores caen, un nuevo servidor sin inicializar no puede convertirse en primario.

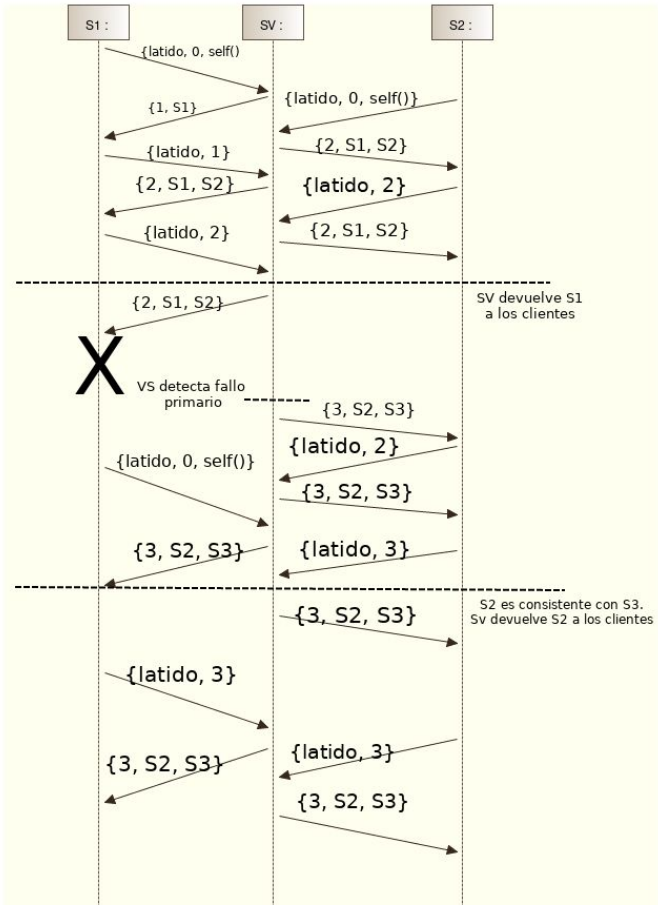


Fig 2. Diagrama de secuencia tras pruebas 1,2 y 3

Las resultados de las pruebas que se han obtenido junto con una traza del estado del servidor se han adjuntado en el Anexo.

Bibliografía

[1] Documentación disponible en moodle.

[2] Ayuda para configurar ssh sin que pida passwords y así poder arrancar las pruebas:
http://www.thegeekstuff.com/2008/11/3-steps-to-perform-ssh-login-without-password-using-ssh-keygen-ssh-copy-id

[3] Funciones y ayuda con Elixir
http://elixirschool.com/

[4] Ayuda con los maps en Elixir
http://elixir-lang.org/getting-started/keywords-and-maps.html

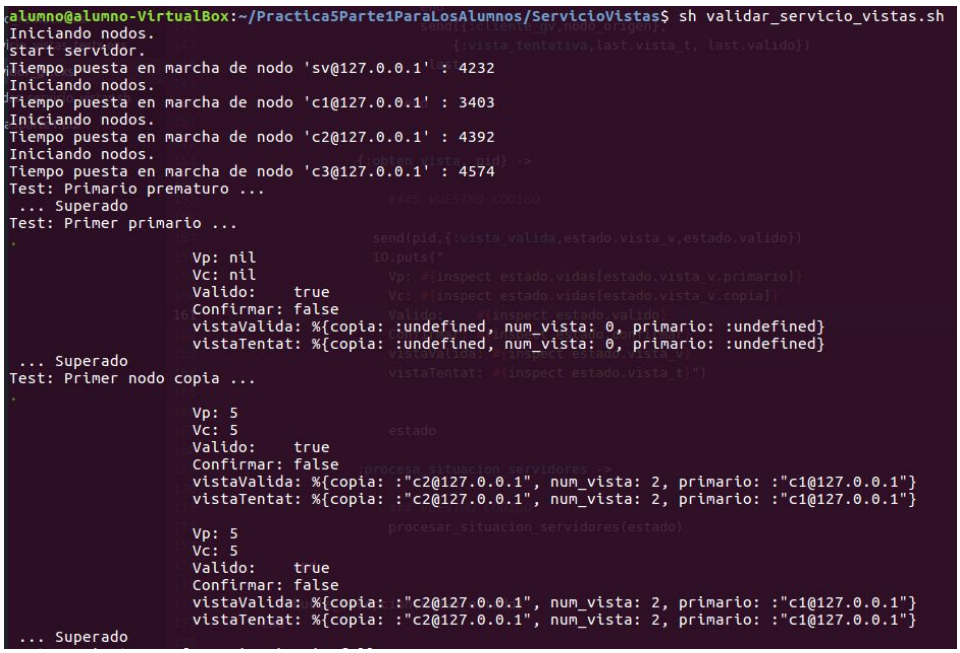
[5] Manejo de mapas en Elixir
https://dockyard.com/blog/2016/02/01/elixir-best-practices-deeply-nested-maps

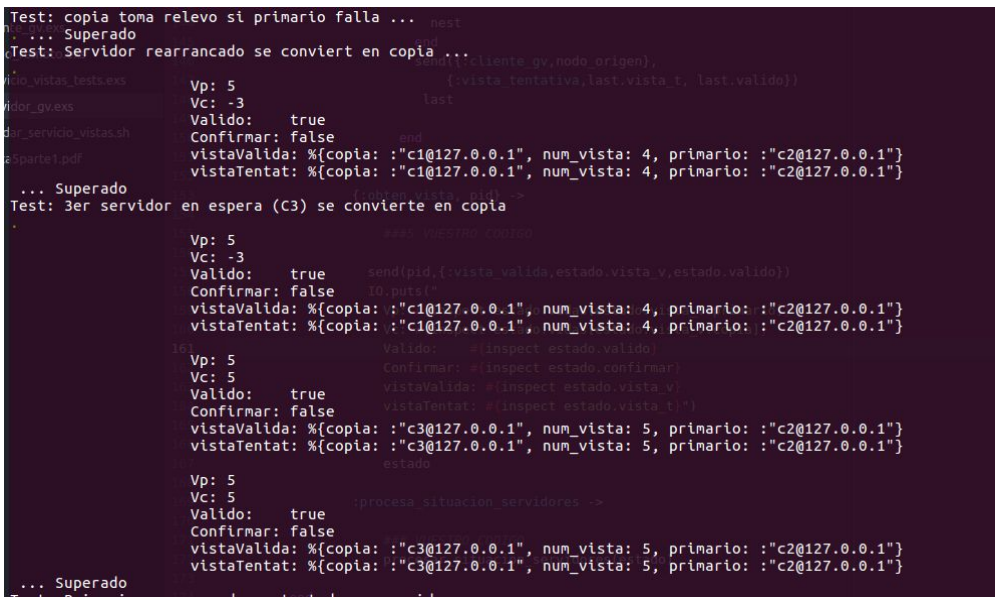
[6] Explicación de los Defstruct
http://culttt.com/2016/06/20/what-are-elixir-structs/

[7] Struts en Elixir
https://www.tutorialspoint.com/elixir/elixir_structs.htm

Anexo

Trazas de las pruebas realizadas con depuración en pantalla, en ellas se muestran los mensajes de depuración usados para testear. En las trazas se refleja el estado cuando es enviado al servidor una petición de obten vista válida, que en las pruebas suelen enviarlas antes de empezar la prueba, con lo que se puede observar la evolución del servidor.





```
Test: Primario rearrancado es tratado como caido
.
                    Vp: 5
                    Vc: nil
                    Valido:     true
                    Confirmar: false
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :"c3@127.0.0.1"}
                    vistaTentat: %{copia: :undefined, num_vista: 6, primario: :"c3@127.0.0.1"}

                    Vp: 5
                    Vc: nil
                    Valido:     true
                    Confirmar: false
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :"c3@127.0.0.1"}
                    vistaTentat: %{copia: :undefined, num_vista: 6, primario: :"c3@127.0.0.1"}
 ... Superado
Test: Servidor de vistas no recibe confirmacion.
.
                    Vp: 5
                    Vc: nil
                    Valido:     true
                    Confirmar: true
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :"c3@127.0.0.1"}
                    vistaTentat: %{copia: :"c1@127.0.0.1", num_vista: 7, primario: :"c3@127.0.0.1"}
 ... Superado
```

```
Test: nuevo servidor sin inicializar
                    no puede convertirse en primario.
.
                    Vp: 5
                    Vc: nil
                    Valido:     true
                    Confirmar: true
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :"c3@127.0.0.1"}
                    vistaTentat: %{copia: :"c1@127.0.0.1", num_vista: 7, primario: :"c3@127.0.0.1"}

                    Vp: nil
                    Vc: nil
                    Valido:     false
                    Confirmar: true
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :undefined}
                    vistaTentat: %{copia: :undefined, num_vista: 6, primario: :undefined}

                    Vp: nil
                    Vc: nil
                    Valido:     false
                    Confirmar: true
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :undefined}
                    vistaTentat: %{copia: :undefined, num_vista: 6, primario: :undefined}

                    Vp: nil
                    Vc: nil
                    Valido:     false
                    Confirmar: true
                    vistaValida: %{copia: :undefined, num_vista: 6, primario: :undefined}
                    vistaTentat: %{copia: :undefined, num_vista: 6, primario: :undefined}
 ... Superado
.Finalmente eliminamos nodos

Finished in 18.3 seconds (0.2s on load, 18.1s on tests)
9 tests, 0 failures

Randomized with seed 0
alumno@alumno-VirtualBox:~/Practica5Parte1ParaLosAlumnos/ServicioVistas$
```