Sistemas Distribuidos I

Trabajo Práctico Final

Alumnos: Kevin Cajachuan (98725), Joaquín Seguí, Joaquín Torré Zaffaroni(98314)

Materia: Sistemas Distribuidos I, 75.74

Cuatrimestre: 2C2019

Profesores: Pablo D. Roca, Ezequiel Torres Feyuk

Introducción

Para el presente trabajo práctico se propone extender la funcionalidad del trabajo práctico 2 incorporando nociones de tolerancia a fallos y manejo de múltiples clientes. El resultado es, entonces, una arquitectura distribuída orientada a *streaming* utilizando *message oriented middlewares* que soporta caídas de los procesos sin que ello afecte al resultado del cómputo.

El diseño de la arquitectura está orientado a un *pipeline* con unidades de cálculo ligadas al negocio, más otros procesos de soporte. En este informe detallamos las decisiones detrás del diseño, documentamos la implementación y marcamos puntos de mejora.

Vista lógica

Esta sección no cambia mucho, es el DAG. Kevin: lo que tenías del TP2 más si hiciste algún cambio.

Vista de desarrollo

Diagrama de paquetes

Vista de proceso

Esquema de multiprocesamiento

Comentar cómo se hizo

Tolerancia a fallos

Para tolerancia a fallos utilizamos un proceso llamado **Watchdog** que recibe los *heartbeats* de los demás procesos, y al detectar que uno se cayó los levanta con la siguiente lógica:

Sin embargo, el **Watchdog** también puede fallar. Debido a eso, tenemos varias instancias del nodo corriendo, solo uno en modo líder y el resto en modo *follower*. Sólo el líder es el que escucha los *heartbeats* y levanta los nodos. Además, el líder manda sus propios *heartbeats* al resto de los *followers*, de manera de que éstos también puedan saber si se muere. En ese caso, se elige un nuevo líder a través del algoritmo Bully.

ídem

Persistencia

ídem

Como el *master* puede caerse, es necesario que otro nodo asuma el rol. Para evitar tener que implementar de nuevo un algoritmo de elección de líder, utilizamos el proceso **Watchdog**. En efecto, cada nodo de almacenamiento agrega metadata a su *heartbeat* con la información del rol que cumple. Esta información es almacenada por el **Watchdog** y esto le permite saber cuándo se cae el nodo maestro.

Entonces es necesario analizar el esquema de replicación teniendo en cuenta las fallas de los nodos. En el caso de un esclavo que se muere y luego se levanta, simplemente puede seguir leyendo de

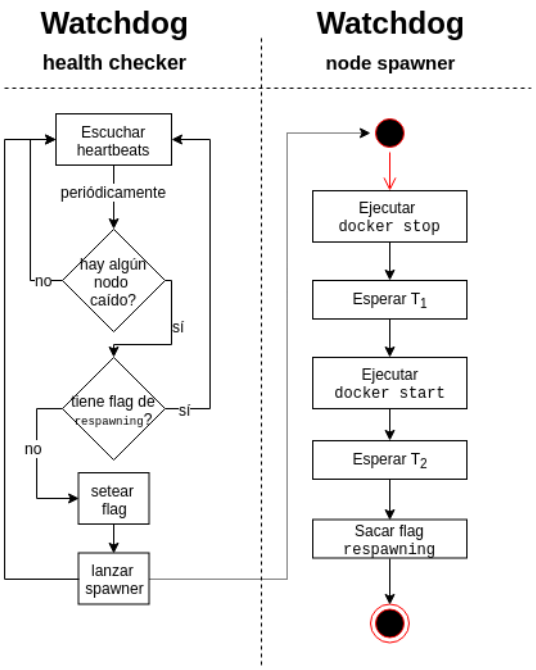

Figure 1:

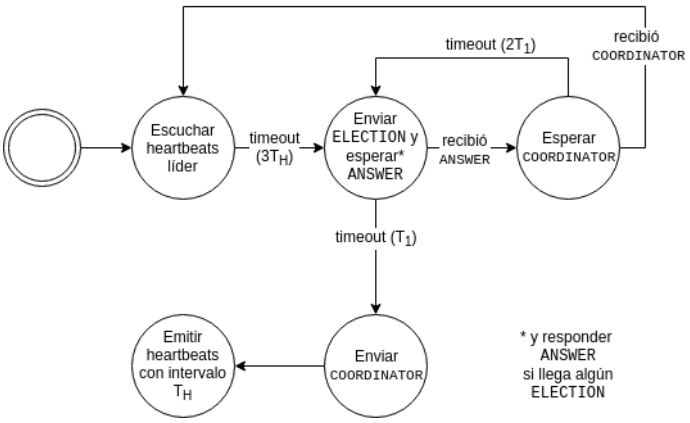

Figure 2:

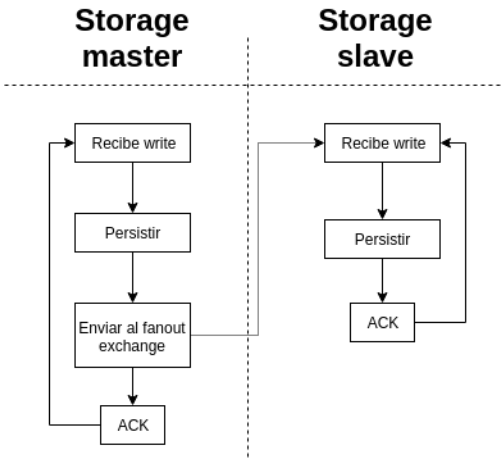

Figure 3:

su cola de Rabbit que es durable. Puede pasar que haya persistido un dato y se haya caído antes de dar el `ACK`, pero como las escrituras son *overwrites* con *timestamps* no es problema. Desde el punto de vista del nodo maestro, si se cae antes de dar el `ACK` el nuevo maestro puede recibir la misma escritura. Por la misma razón que antes, esto no es problema.

A continuación se puede ver el diagrama que ilustra la generación de un nuevo nodo maestro.

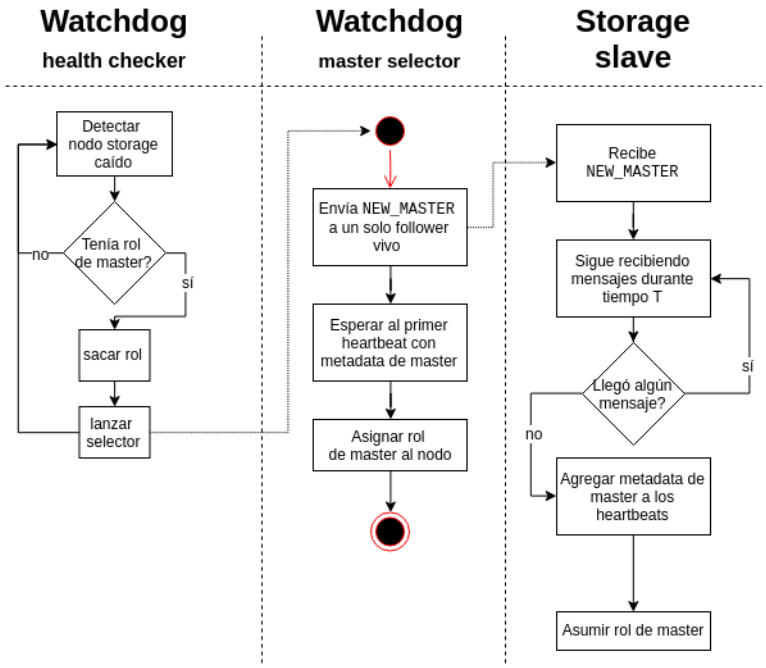

Figure 4:

Vista física

Diagrama de despliegue

Escenarios

Comentar los casos de prueba que tenemos en el doc de aceptación

Conclusiones

Puntos de mejora

Referencias

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). Distributed systems: concepts and design. pearson education.

Apuntes de clase.