SISTEMAS DISTRIBUIDOS

PRÁCTICA 3:

Tolerancia a fallos en Servidores Sin Estado

Alejandro Terron 761069
Óscar Anadón 760628

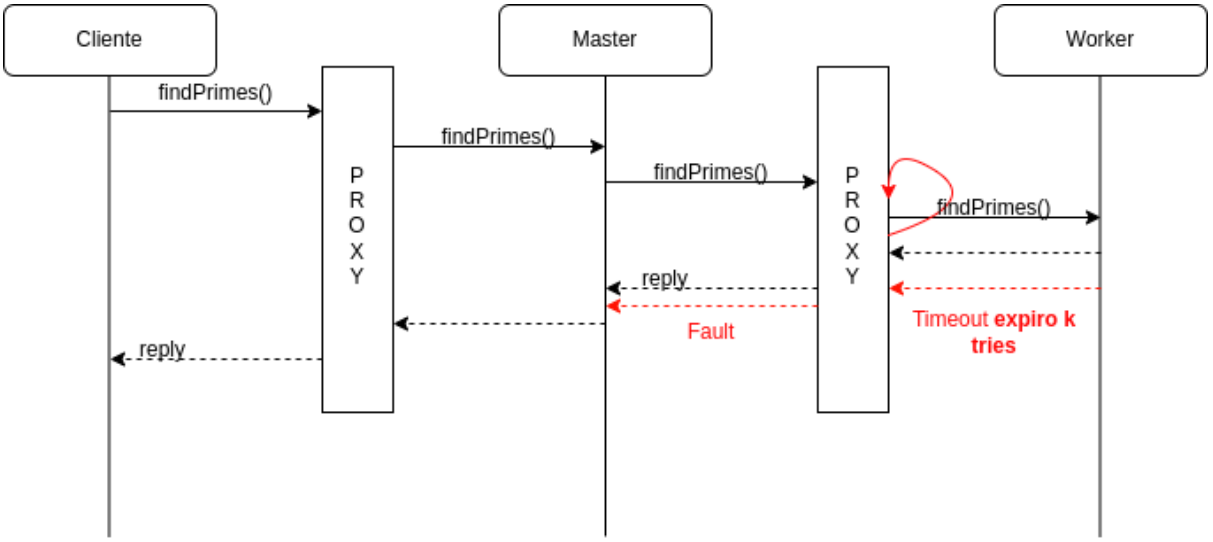
19-noviembre-2021

RESUMEN

Para esta práctica se ha planteado un problema en el cual se ha tenido que diseñar sobre una arquitectura master worker similar a la de la práctica 1, pero esta vez la comunicación entre los distintos nodos del sistema se ha realizado mediante el protocolo RPC además de tener que implementar técnicas de tolerancia a fallos.

Para ellos se nos proporciona el generador de la carga de trabajo (cliente) y el código del Worker el cual no ha sufrido ninguna modificación. Como base se escogió el código master perteneciente a la práctica 1.

Para poder invocar la llamada a la función FindPrimes desde el cliente a un worker ha sido creada una función homónima en el master.El FindPrimes en el master se encarga de recibir las distintas peticiones de los clientes, devolver los resultados proporcionados por los workers y pasarle los datos a traves del proxy entre es master y los workers.



En la función interacción() es donde se ha implementado la tolerancia a fallos para:

- Los fallos de tipo crash: son los más fáciles de reconocer ya que al realizar la invocación remota a la función se produce un error, este tipo de fallo es incorregible por lo que se ha decidido a cerrar el proxy correspondiente evitando asi resolver peticiones mediante workers inactivos.
- Los fallos de tipo timing/delay: la respuesta se demora más de los esperado, para estos se ha implementado un select el cual se resuelve cuando el worker devuelve la respuesta o salta el timeout establecido, caso en el cual se procede a duplicar el timeout y volver a invocar a la función.
- Los fallos de tipo omission: A pesar de lograr realizar la invocación remota no se recibe respuesta alguna, el mecanismo anterior de los timeout sumado a un número máximo de intentos sirve para detectar este tipo de errores.

Una vez se tenia el funcionamiento básico del sistema mediante rcp y tolerancia a fallos se procedió a la implementación del algoritmo del matón, el cual se basa en el lanzamiento de un conjunto de procesos entre los cuales uno (el que presenta mayor id) ejerce la figura de coordinador, en nuestro caso es un conjunto de masters que permanecen esperando hasta que el algoritmo les da la figura del coordinador. Si el coordinador cae los demás procesos se dan cuenta bien porque no les llega el latido del coordinador por lo cual empiezan un proceso de elección que propagan a aquellos procesos con masyor id que ellos o bien porque les llega un mensaje de otro proceso con menos id pidiendo una elección al cual responden con un ok. Este procedimiento finaliza cuando el proceso con mayor id de los restantes se da cuenta que es el nuevo coordinador, propagando así otra cadena de mensajes a todos los nodos informando que él es la nueva figura del coordinador. Con esto conseguimos masters alternativos que seguirán realizando la misma función y de cara al cliente, este proceso es totalmente transparente (aunque siempre habrá un delay ligeramente superior para dar tiempo a que el nuevo máster esté listo para servir peticiones).

PRUEBAS

Como se conoce el QoS el cual se tiene que conseguir se ha procedido a realizar 3 pruebas con divesos timeouts calculados para no violar el QoS, todos ellos con la misma carga de trabajo y el mismo numero de intentos, 3. En la **Figura 1** se representa la prueba en la cual se parte de un timeout de 425 ms y se duplica por intento, como se puede ver aparecen algunas de las tareas sobre el QoS eso quiere decir que no han podido ser resueltas en tiempo.

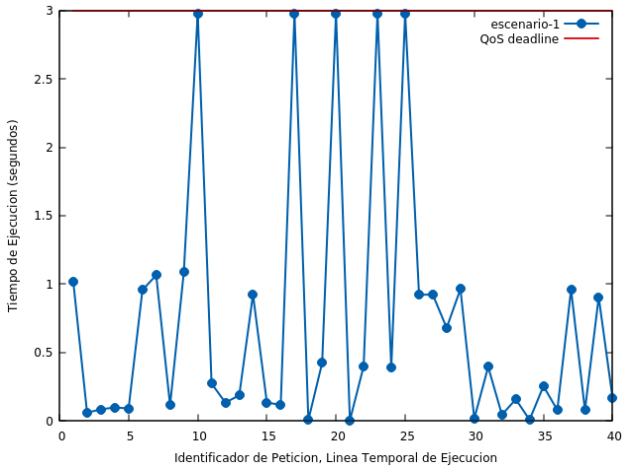


Figura 1: Duplicando timeout

La **Figura 2** corresponde a la prueba aumentando el timeout en función del intento, así pues en el primero es de 750 ms, en el segundo de 1500 ms (750*2) y en el tercero de 2250 ms (750*3).

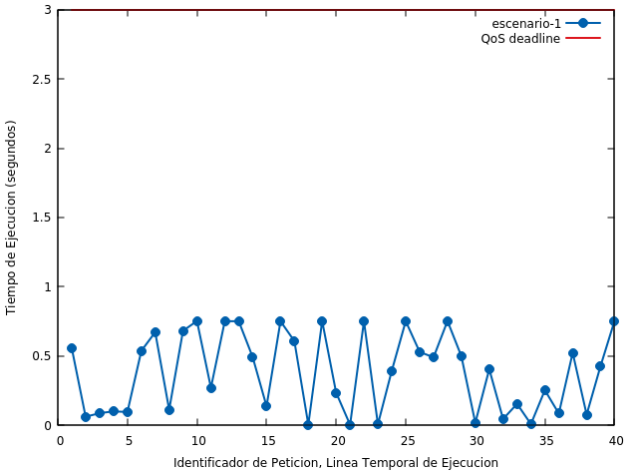


Figura 2: Timeout * Retry_number

En la **Figura 3** se encuentra representada la prueba con timeout figo de un segundo

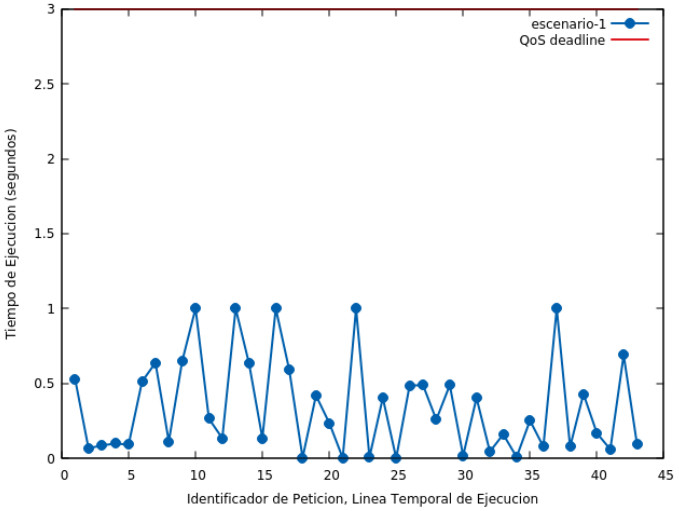


Figura 3: timeout fijo (1 segundo)

Como se puede ver al conocer el QoS es fácil imponer fórmulas para que el proceso no lo viole ya sea porque ha resuelto la petición o ha agotado el tiempo, a priori las mejores son las representadas en la Figura 2 y 3, pero al generarse los errores de manera aleatoria no se puede saber a ciencia cierta.

Conclusión

Con esta práctica hemos conseguido entender varios puntos vistos en la asignatura como son la tolerancia de fallos de tipo crash, omisión y timing, el funcionamiento de los proxys y la invocación remota de funciones mediante el uso de RPC. Esto es importante ya que a partir de esto se incorporan medidas que garanticen el QoS para cada una de las tareas que solicite el cliente. En cuanto al algoritmo del matón nos ha parecido bastante interesante y útil, ya que en el mundo real no se puede suponer que un nodo nunca vaya a finalizar de forma anómala.