Sistema Distribuido tolerante a fallos

Sergio Herrero Barco 698521

Alex Oarga Hategan 718123

Noviembre de 2017

1. Introducción

El sistema diseñado consiste en un Sistema Distribuido tolerante a fallos, que permita encontrar todos los números amigos de un intervalo. Dos números a y b se dicen que son amigos si y solo si la suma de los divisores de “a” es igual a “b”, y la suma de los divisores de “b” es igual a “a”. Para ello se va a usar una arquitectura Master-Worker en la que se va a implementar un algoritmo de detención de fallos.

2. Arquitectura

Se han creado tres ficheros en los que esta implementado dicho sistema. Un fichero “amigos.exs” que contiene las operaciones relacionadas con los números, otro fichero “worker.exs” que contiene el módulo de los worker, y por ultimo un fichero “main.exs” que contiene el modulo del master.

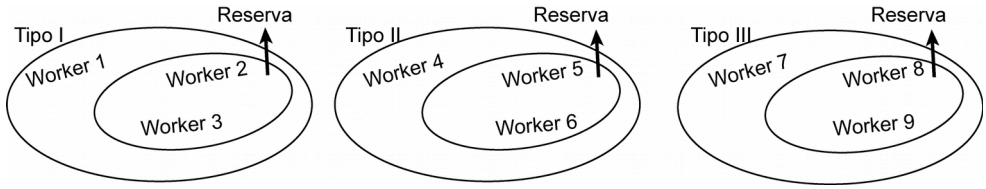
2.1. Arquitectura del Sistema

El sistema se compone de una serie de nodos que van a actuar siguiendo una Arquitectura Master-Worker. Dicha arquitectura consiste en un reparto de tareas, el master solicita operaciones al worker y este se las devuelve resueltas.

Existen 3 tipos de workers:

1. Los Worker de tipo 1 resuelven la tarea de encontrar los divisores de un número.
2. Los Worker de tipo 2 resuelven la tarea de sumar la lista de divisores de un número.
3. Los Worker de tipo 3 resuelven la tarea de sumar los números de una lista.

Cada tipo de worker **alberga 3 réplicas**, siendo una la líder, con la misma funcionalidad y así poder implementar un sistema de detención de fallos.

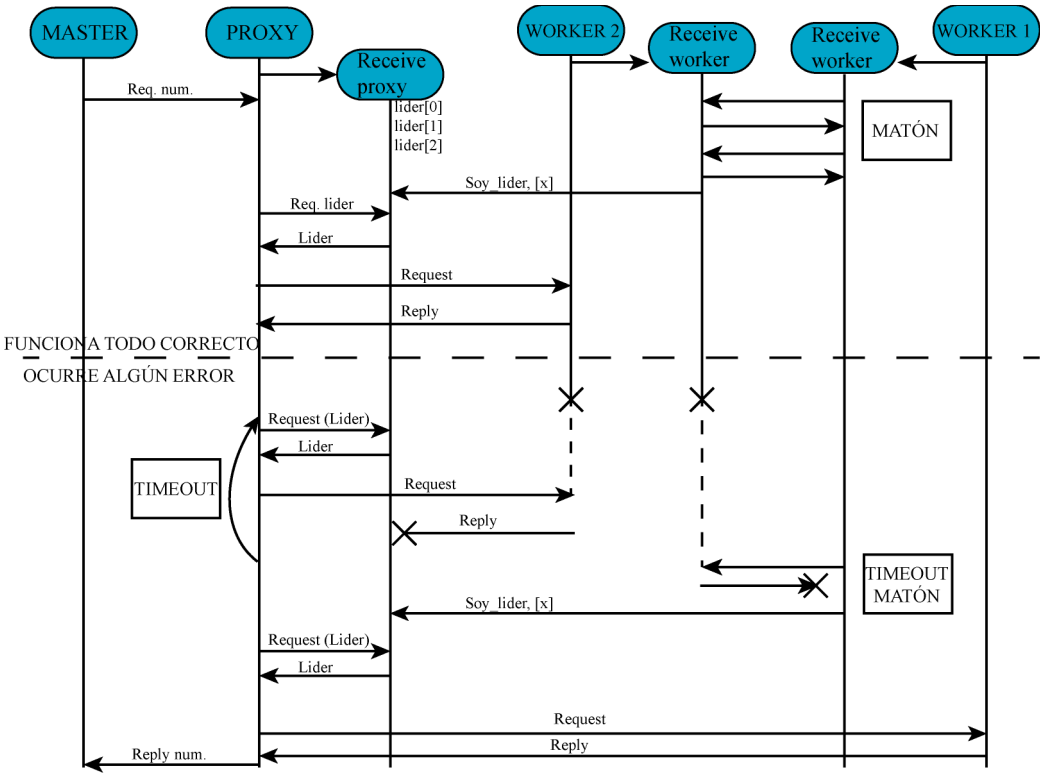


El **master** no se comunica directamente con los worker, sino que tiene implementado otra arquitectura Master-Worker que reparte la tarea entre dos procesos llamados **proxy**, y estos son los encargados de enviar las peticiones a los diferentes workers.

El **proxy** realizara peticiones a los worker y activara un temporizador esperando la respuesta. Si la respuesta no llega en un tiempo, vuelve a realizar la petición.

Los **worker** a la vez que responden a las peticiones que le llegan, tiene implementado el algoritmo del **matón**. Si en algún momento el worker lider de un tipo sufre un fallo, se activa una encuesta para elegir el nuevo líder.

2.2. Protocolo de comunicación



3. Tolerancia a fallos

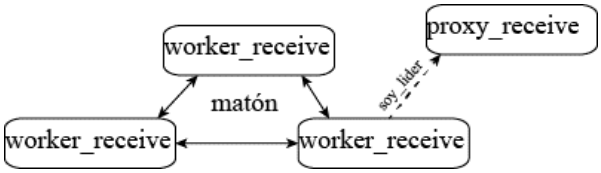
3.1. Master

El master tiene dos factores para tratar la tolerancia a fallos. El primero es la implementación del algoritmo 1 de tolerancia, que consiste en tener un servicio proxy que se comunica con los worker. Este le envía una petición, y si pasado un tiempo no contesta, vuelve a enviar la petición. Si tras 5 intentos no se recibe respuesta, el proxy envía un mensaje al master indicándole que el worker no funciona.

Además del algoritmo, el master envía primero la tarea al worker 2, y si este falla, le envía la petición al worker 1 y al worker 3. Siempre se le da preferencia al worker 2 ya que es el único que realiza la operación completa.

3.2. Worker

El worker implementa el algoritmo del matón. Dicho algoritmo consiste en la detección de que un nodo “líder” sufre una caída, y la elección de un nuevo líder que se ocupe de la funcionalidad. Tras la elección de un nuevo líder, este es el encargado de avisar al **proxy** de que es el nuevo líder y es el que va a realizar la funcionalidad.



4. Validación experimental

4.1. Timeout

Tras la ejecucion del codigo desarrollado se ha pasado a ajustar los timeouts que se han utilizado. Tanto para el algoritmo del maton como para el algoritmo del proxy el objetivo es ajustar el timeout al tiempo 2RTT + Tprocesamiento.

En el algoritmo del maton, la media del tiempo que se ha obtenido en la ejecucion es de 3 ms. En la implementacion se ha decidido dejar un margen al tiempo y se ha declarado un timeout de 30 ms.

En el algortimo del proxy como el timeout debe ser mayor que el tiempo de ejecucion del algoritmo del maton para poder detectar los fallos a tiempo se ha utilizado un timeout de 50 ms.

4.2. Número de workers

En un principio se han planteado 3 workers de cada tipo. Tras la ejecución se va a plantear el número de workers óptimo de cada tipo.

La media del tiempo que lleva el calculo de los divisores de un número es de 9,12 segundos. Este resultado es normal teniendo en cuenta que en el 75% de los casos los workers tienen un error de timing. A su vez, como cuando los workers 2 tienen un error se recurre a los workers de tipo 1 y 3, la relación de que a obtenido de veces que se recurre a cada worker y este no presenta fallos es de: 5/6 worker de tipo 2, 1/6 worker de tipo 1 y 3. A partir de esto se obtiene que la disposición en número de workers debe ser como la anterior proporción.

Teniendo en cuenta el tiempo de ejecución del worker, (2RTT + Tproc) el número de workers para minimizar el tiempo en coste de los fallos deberia ser: 9.12/ (2RTT+Tproc).

5. Conclusión

Se ha creado una arquietectura master-worker con tolerancia a fallos. Para los fallos en el master, se ha implementado un proceso proxy con un timeout para la comunicación con los workers. Para la deteccion de errores entre los workers se ha desarrollado el algoritmo del maton para la comunicación entre workers del mismo tipo. Se ha planteado también un patrón master-worker para el reparto de la carga entre el master y los proxys.