Práctica 3.
Números amigos

Andrés Gavín Murillo 716358

Borja Aguado Díez 741440

23 de noviembre de 2018

Introducción

Se ha desarrollado un sistema con una arquitectura de tipo Master/Worker, en el cual existen 3 tipos de workers para realizar 3 tipos de operaciones distintas. Estos Workers tienen una probabilidad de sufrir un fallo de distinto tipo. El objetivo del sistema es encontrar los pares de números amigos que se encuentran dentro de un intervalo predeterminado.

Sección Principal

Resumen

Tanto el Máster como los Workers se han desarrollado en Elixir. Debido a que los Workers son susceptibles a fallar, se ha implementado el algoritmo de detección de fallos enseñado en clase [1]. Este algoritmo es capaz de detectar los tres tipos de fallos posibles (crash, omission, timing) a la hora de poner en marcha el sistema.

El Máster se comunica con los workers para que realicen determinadas operaciones, y calcula los pares de números amigos en el intervalo con la ayuda de ciertas funciones internas que se explican en el siguiente apartado.

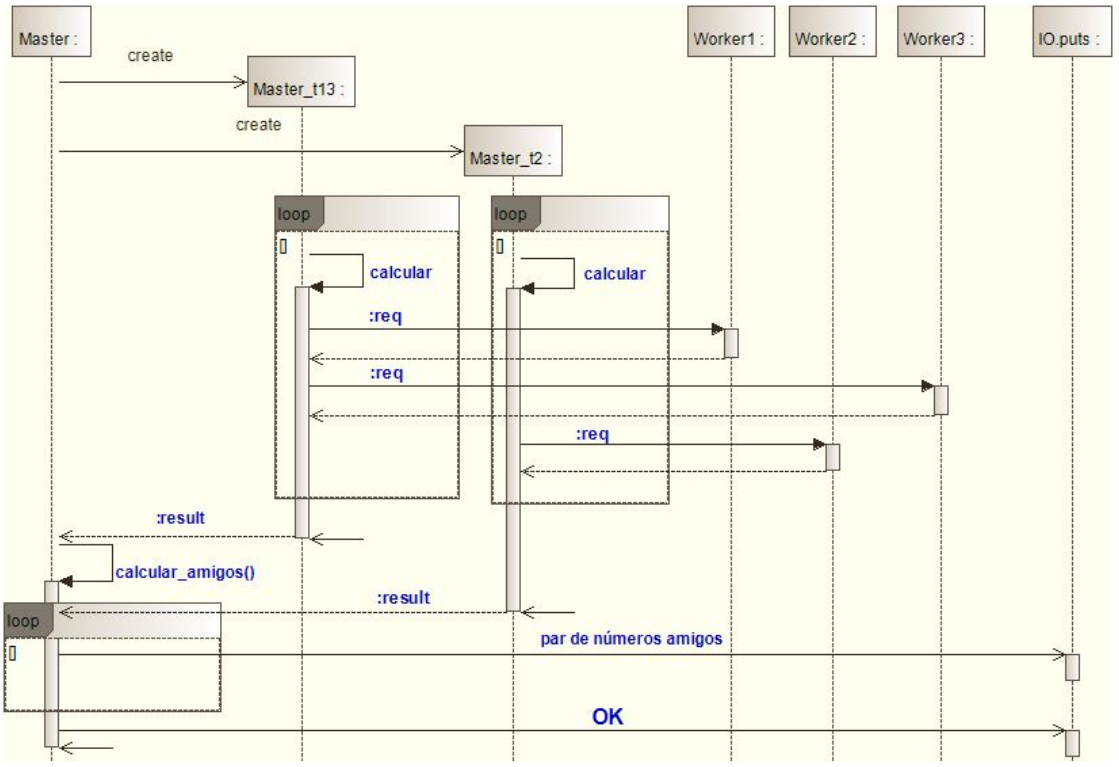
Arquitectura del sistema

El Máster se ejecuta en una máquina, y los Workers ocupan tantas máquinas como sea necesario. El Máster tiene la lista de los 3 tipos de nodos Worker que se encuentran conectados a la red.

El proceso principal del Máster lanza 2 procesos, uno se encarga de las operaciones 1 y 3 y otro se encarga de la operación 2. Su función es calcular lo que se le pide a través de los Workers, y finalizar los cálculos cuando se lo indiquen (debido al timeout, por el tratamiento de errores). Para ello, invocan repetidamente a la función calcular, que es la que se comunica directamente con los workers. Esta función también se encarga de comprobar que respondan dentro del timeout y de reintentar en caso contrario. Una vez terminados los cálculos, los devuelven al proceso principal.

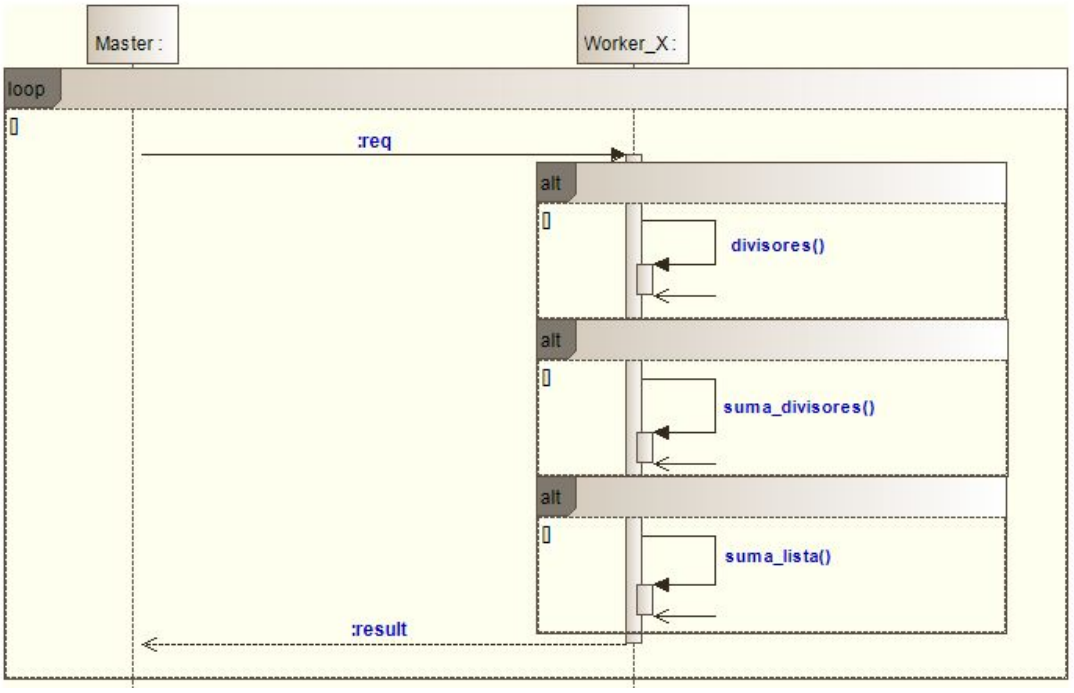
Por último, se invoca a la función calcular_amigos, que utiliza una función de inmersión para comprobar e imprimir por pantalla los pares de números amigos en el intervalo.

A continuación se muestra el funcionamiento en un diagrama de secuencia. El último elemento corresponde a la salida estándar; se ha añadido para visualizar temporalmente cuando el programa imprime por pantalla.

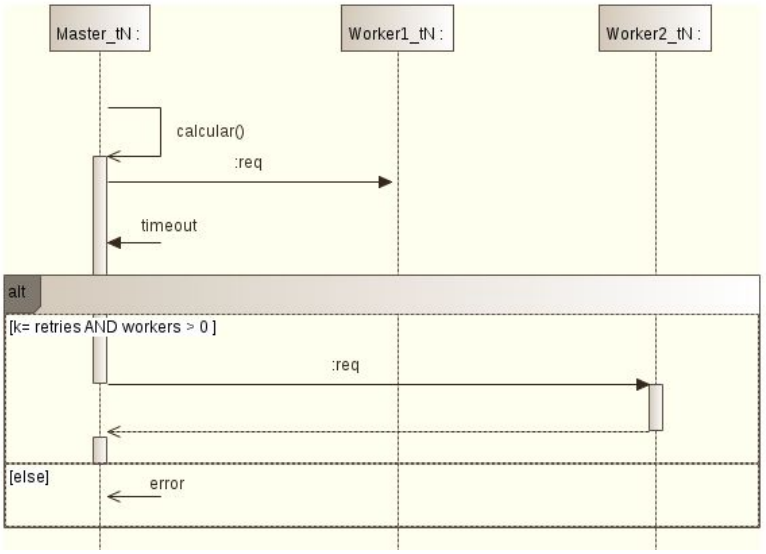


En cuanto a los Workers, al inicializarse tienen una cierta probabilidad de presentar algún fallo (crash, omission, timing). Una vez simulado el fallo, y si no se ha producido un fallo de tipo crash (en cuyo caso se quedaría bloqueado en un bucle infinito) queda a la espera de recibir una petición del Máster, y cuando la recibe, la procesa y devuelve el resultado.

A continuación se muestra el funcionamiento en un diagrama de secuencia.



En caso de que el worker presente algún fallo, y no se hayan agotado las “retries” ni los workers disponibles, se comunica con el siguiente worker:



Validación

El sistema funciona correctamente, para su validación ha sido probado con éxito en las siguientes situaciones:

- Fallos de los workers deshabilitados y un worker de cada tipo.
- Fallos de los workers habilitados y diez workers de cada tipo.

El sistema va mostrando, según los calcula, los pares de números amigos entre el 1 y el 1.000.000.

Conclusiones

El tratamiento de errores es una parte crucial a tener en cuenta a la hora de desarrollar un sistema distribuido. Muchos algoritmos dependen de que no se de ningún fallo cuando el programa se está ejecutando, y lamentablemente eso dista de ser el caso en la práctica.

Una buena implementación de algoritmos para el tratamiento de fallos en el sistema ofrece una cierta garantía de que este podrá seguir funcionando si los nodos son propensos a fallar o la red es inestable, entre otros escenarios.

Referencias

[1] Diapositiva número 35.
https://moodle2.unizar.es/add/pluginfile.php/1876820/course/section/305077/L04-Fault-Tolerance.pdf?time=1540307419788