Práctica 3
Tolerancia a fallos

César Martín Génova - 649711

José Félix Longares Moreno - 696215

16 de noviembre de 2017

Introducción

En esta práctica 3, se plantea el problema de crear un sistema distribuido que siga una arquitectura Master-Worker, teniendo en cuenta que la ejecución de los workers está sujeta a fallos de tipo crash, omission o timing, que dado un intervalo de números, del 1 al 1000000, encuentre los pares de números amigos. Se dice que dos números son amigos si los divisores propios del primero suman el segundo número y viceversa.

A lo largo de la memoria se explicará la implementación del algoritmo además de los problemas y soluciones que se han encontrado a la hora de realizar el diseño en Elixir.

Implementación

Se plantea la creación de dos archivos .exs, el primero contendrá el desarrollo del programa principal y el segundo contendrá las operaciones necesarias para la comprobación por parte de los workers de los pares de números de posibles amigos y del master para comprobar los resultados obtenidos de los workers.

El archivo operaciones contiene las operaciones de búsqueda de divisores, la suma de los mismos y la comprobación de la amistad entre ellos.

El archivo principal contiene tres módulos, el módulo Master, el módulo Worker y por último el módulo proxy.[1][2]

El módulo Master comienza creando un hilo en el que ejecuta el proxy, y a continuación espera a que los workers se registren, una vez que estos se han registrados, llama a la función comprobar, pasándole el 1, el primer número a estudiar, una lista vacía en la que se introducirán los números visitados y el pid del proxy.

La función comprobar, compara el número que le ha sido pasado con los introducidos en la lista de visitados, en caso de que esté, se llamará a si misma con el siguiente número, en caso de que no esté llamará a la función calcular amigos. Esta función recibirá como parámetros el número a estudiar, el pid del proxy y la lista de visitados. Automáticamente esta función llamará a bucle2, pasándole como parámetros el número a estudiar, un 0 como segundo número y un 0 como suma de los divisores del primero, estos se actualizarán conforme avance la ejecución, un entero v que puede tomar valores 1 o 0 dependiendo si está estudiando el segundo o el primer número de la pareja de posibles amigos y el pid del proxy.

Bucle 2 mandará un request al proxy pasándole el número a estudiar, el pid del master y el tipo de trabajador que se busca, el proxy en consecuencia creará un hilo que ejecutará la función action y le pasará a esta como parámetros el número, el tipo de worker, el pid del worker y del master el tiempo de timeout y el número de reintentos y a continuación se vuelve a relanzar el proxy.

La función action envía al pid del worker una request con su pid y el número a estudiar, y este le devuelve el resultado de ejecutar la función op que se encuentra dentro del

módulo worker, dependiendo del tipo del worker realizará si es tipo 1 una búsqueda de divisores propios que introducirá en una lista, si es de tipo 3 sumará dicha lista de divisores y si es de tipo dos realizará ambas operaciones.

En caso de que el timeout venza, volverá a llamarse la función action a si misma actualizando el número de reintentos, una vez llegado a 5 enviará al master un mensaje de error. En caso contrario, enviará al master la solución encontrada.

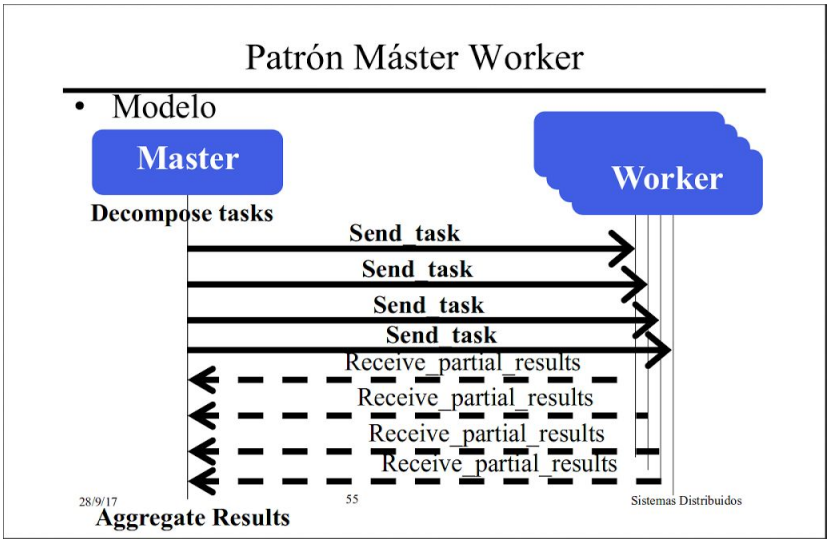
Volviendo al bucle2 existe la posibilidad de recibir la solución o un error. Si se está estudiando el primer número y se recibe la solución, se llamará de nuevo a bucle2 pidiéndole que realice la comprobación del segundo número y actualizando la lista de visitados, si en cambio se está comprobando el segundo número, llamará a la función sonAmigos y en caso de que lo sean se mostrará por pantalla y a continuación se llamará a la función comprobar con el siguiente número a buscar y actualizando la lista de visitados. En caso de error en primera vuelta, se llamará a la función bucle1y3 para que estudie el primer valor, en caso de que sea en segunda se llamará a la función bucle1y3 para que realice el estudio correspondiente.

Bucle1y3 tiene un funcionamiento similar a bucle2, puede recibir también una solución o un error, si recibe la solución y está en primera vuelta enviará al tercer tipo de worker una petición para que realice la suma de los divisores propios, esto puede fallar, que, en este caso, se llamará a bucle2 con los parámetros iniciales o que te devuelva las soluciones, en cuyo caso, se llamará a bucle2 para que compruebe el segundo número y se añadirá el primero a la lista de visitados.

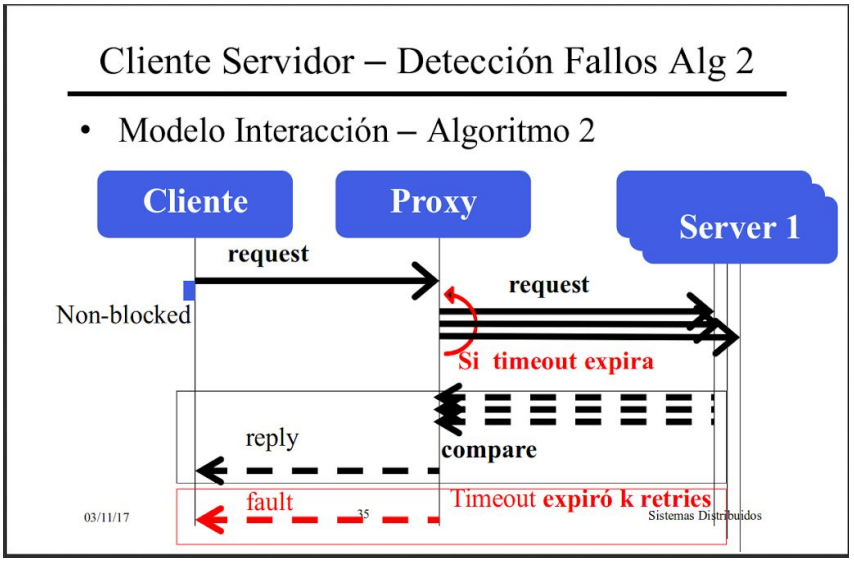
En segunda vuelta si se ha recibido la solución del worker1, la lista de divisores propios, se llamará al tercer worker para que nos devuelva la suma, si nos devuelve la solución, se comprobará si son amigos, si lo son se mostrarán por pantalla, se añadirá a la lista de visitados el segundo número y se llamará de nuevo a la función comprobar con el siguiente número a estudiar, si da error se llamará a bucle2 pasándole el segundo término a estudiar.

En caso de que recibamos un error del worker1 si se encuentra en primera vuelta se llamará a bucle2 con los parámetros iniciales y en caso de que se encuentre en segunda, exactamente igual pero con los parámetros del segundo término.

Una vez que comprobar llegue a 1000000 mostrará por pantalla un mensaje indicando la finalización del programa.



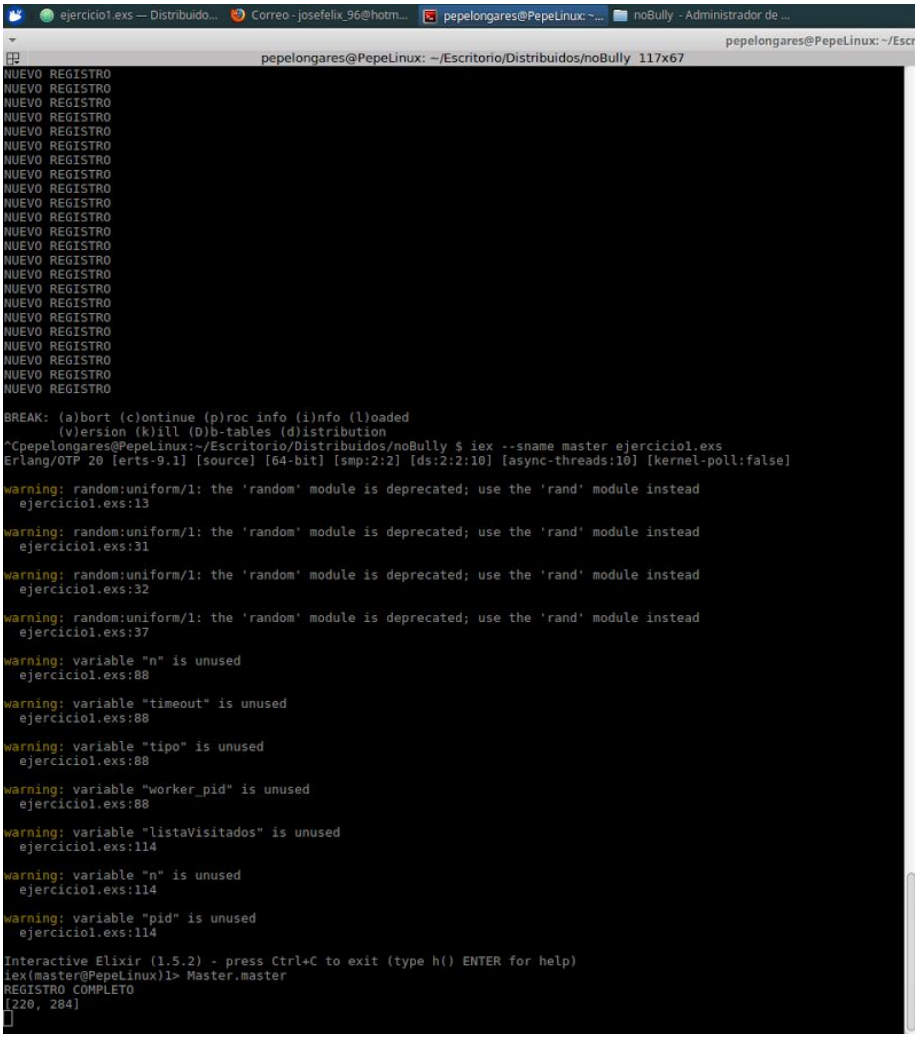
Esquema del sistema implementado en la práctica, sin tener en cuenta el proxy



Sistema de detección de fallos implementado en la práctica

Validación Experimental

Como podemos observar en la imagen, se encuentra la primera pareja de números amigos, sin encontrar ninguna “errónea” por el camino. Debido al tiempo de ejecución tan alto necesario para encontrar más parejas, sólo se muestra el programa a la hora de encontrar la primera de ellas.



Conclusiones

El tratamiento de errores es una de las partes más complicadas de los sistemas distribuidos; incluso llegando a convertirse en algo imposible de controlar, ya que determinados fallos como los bizantinos no tienen un tratamiento claro ni una manera de detectarlos. El tratamiento de los errores implementado es uno de los más sencillos, en el que sólo se detecta, pero no se arregla dicho fallo.

Bibliografía

[1] Arronategui Unai, Tolosana Rafael, “Introducción al lenguaje Elixir” Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 26 Sept 2017

[2] Elixir web https://elixir-lang.org/