SISTEMAS DISTRIBUIDOS

Práctica 1

Ignacio Palacios Gracia 739359
Rubén Rodríguez Esteban 737215

Índice de contenidos

1. Resumen **3**

2. Introducción **3**

3. Primer escenario **3**

4. Segundo escenario **4**

5. Tercer escenario **5**

6. Cuarto escenario **6**

7. Validación **7**

8. Conclusión **8**

9. Referencias bibliográficas **8**

1. Resumen

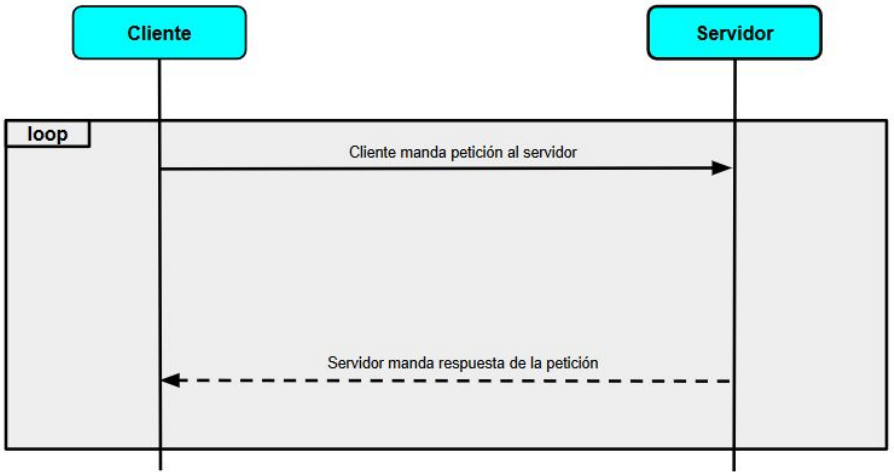
En esta memoria se explica las diferentes arquitecturas empleadas para un conjunto de escenarios en los que un conjunto de procesos se comunican intercambiando mensajes. Para cada escenario se describe por qué se ha empleado esa arquitectura y se realizan un conjunto de mediciones de tiempo acerca del diseño.

2. Introducción

Esta práctica ha consistido en la realización de un conjunto de escenarios, concretamente cuatro, en los que se pide devolver una lista de cuáles son los primeros números perfectos en los 10000 primeros números naturales, utilizando una arquitectura distinta para cada uno de los escenarios planteados en el guión de la práctica. A continuación se describen cuáles son los escenarios realizados junto con sus correspondientes arquitecturas de diseño junto con las justificaciones de por qué se ha optado por dicho tipo de arquitectura.

3. Primer escenario

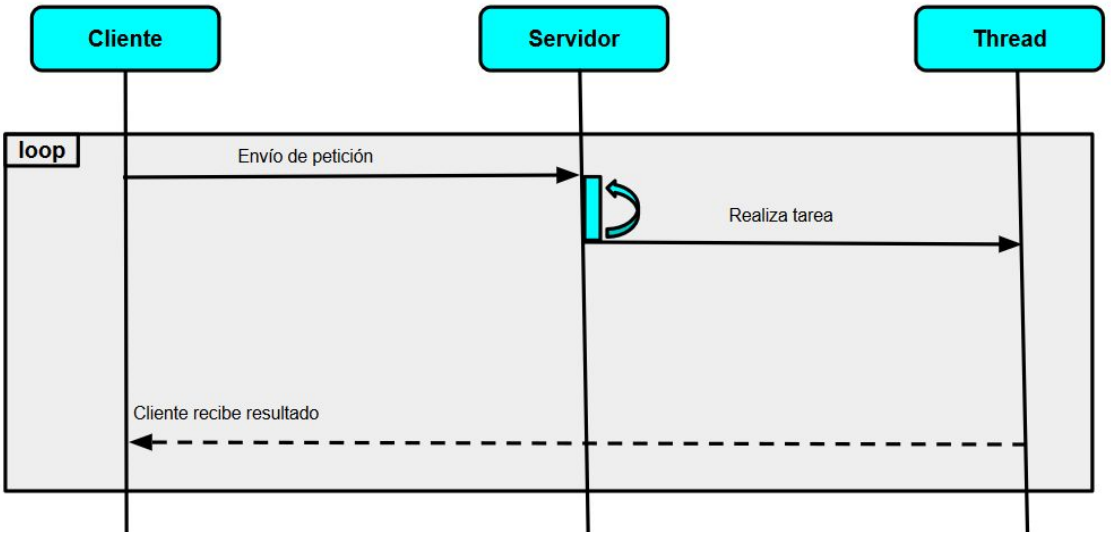
Este escenario tiene como condiciones que el cliente realiza una petición cada dos segundos. Para llevar a cabo este escenario se ha empleado la arquitectura de comunicación síncrona. En este escenario han intervenido únicamente dos procesos, un proceso cliente que envía peticiones y un proceso servidor que las atiende, y una vez finalizadas, envía la respuesta al cliente. El paso de mensajes entre ambos procesos se efectúa tal y como muestra el diagrama de secuencia siguiente:



La principal razón por la que se ha escogido este mecanismo arquitectural es que hay muy pocas peticiones de trabajo al servidor por parte de los clientes pudiéndose asegurar que cuando se produce una nueva petición del cliente al servidor, la petición que el cliente efectuó anteriormente ya ha finalizado o está casi para finalizar. Por lo que la comunicación entre ambos es casi instantánea.

4. Segundo escenario

El cliente realiza más de una petición cada dos segundos, dependiendo del sistema. En este escenario se ha optado por usar una arquitectura de comunicación basada en un servidor concurrente. Participan un proceso cliente que solicita las peticiones y un proceso servidor que por cada tarea que recibe del cliente lanza un thread que la ejecuta. El paso de mensajes se realiza tal y como muestra el diagrama de secuencia siguiente:

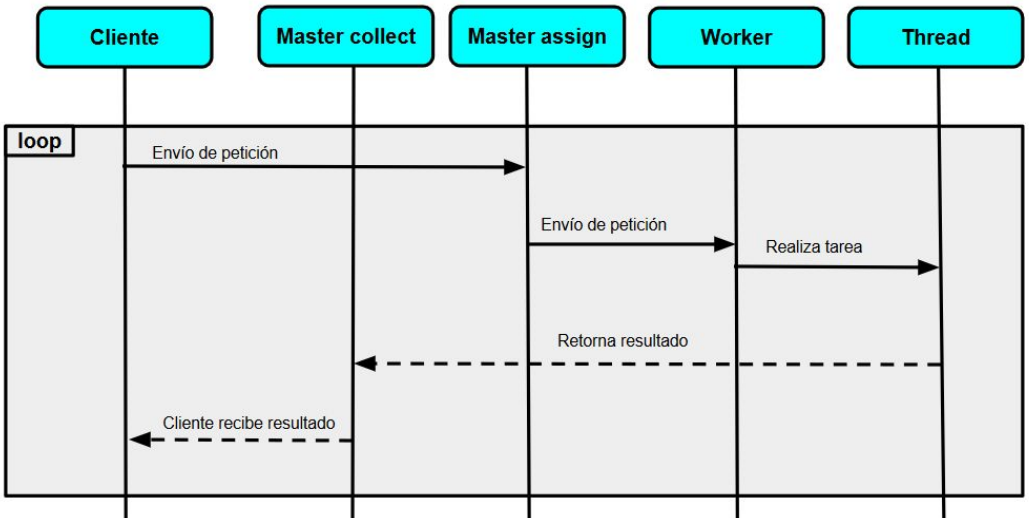


La razón principal por la que se ha usado esta arquitectura es porque aunque es un intervalo de tiempo pequeño, el servidor tiene los recursos suficientes para poder atender todas las peticiones de los clientes.

5. Tercer escenario

El cliente realiza más de una petición cada dos segundos, dependiendo del sistema.

Para llevar a cabo este escenario se ha optado por escoger una arquitectura master-worker. En este escenario intervienen un proceso cliente, un proceso master subdivido en dos hilos o threads y una lista de workers. La comunicación entre los procesos anteriores se realiza tal y como muestra el diagrama de secuencia siguiente:

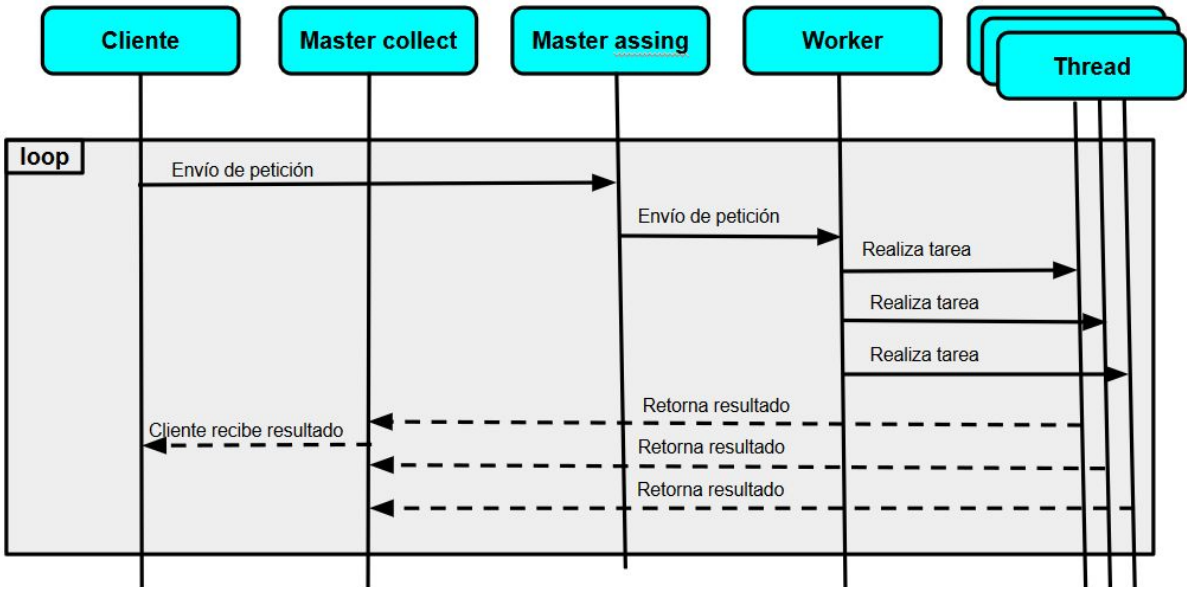


En este escenario el cliente le manda una petición al proceso master, subdivido en dos hilos o threads de tal forma que uno de los hilos del máster (Master assign) atiende la petición del cliente y se la manda a un proceso worker de una lista. Una vez que el worker termina el cálculo, le manda el trabajo realizado al otro thread del máster (Master collect), y este se lo manda al cliente. Para poder ejecutar correctamente la arquitectura, sin ninguna violación del QoS, se han necesitado al menos 3 workers conectados al máster. El número ha sido determinado por prueba y error, en el que pudimos comprobar que no se cumplía el QoS con un número más bajo.

6. Cuarto escenario

El cliente realiza más de una petición cada dos segundos, dependiendo del sistema y además los recursos computacionales tienen cierto grado de indeterminismo:

Para realizar este escenario se ha diseñado una arquitectura de master-worker. Los procesos que intervienen en esta arquitectura son un cliente que envía las peticiones, un proceso master que las recibe y se las asigna a distintos workers, y una lista de procesos workers que se encargan de realizar las peticiones. El envío y recepción de mensajes entre los procesos ha sido gestionada como se muestra a continuación.



En primer lugar el cliente le manda una tarea al master. Una vez que el master recibe la petición del cliente le asigna un identificador a dicha tarea expresada por un dígito entero, y se la envía a un worker. Dicho worker creará diferentes threads, todos ellos con la misma tarea. De esta forma, aunque cada thread por separado tenga una alta probabilidad de dejar de funcionar por un tiempo, esta probabilidad se reduce al utilizar al menos 3 threads diferentes. Mientras los threads están trabajando, el máster espera a que uno de ellos, el más rápido de todos, le envíe un mensaje con la respuesta. Una vez que el worker le manda la respuesta al máster, éste último se la envía al cliente que solicitó la petición procediéndose a atender la siguiente. De este modo, los threads que quedan por enviar su respuesta la enviarán pero serán ignorados por el master dado que se está atendiendo las peticiones siguientes. El máster controlará aquellas peticiones ya atendidas mediante una lista, en la que se almacenan aquellos identificadores de tareas que ya se han atendido y podrá ignorarlas, además de aquellas peticiones con un identificador de tarea más bajo que la tarea esperada actual.

7. Validación

A lo largo de este trabajo, más en concreto en los dos últimos escenarios, se ha utilizado principalmente un método de prueba y error para averiguar cuántos elementos (workers, threads) necesita la arquitectura escogida para poder funcionar sin problemas.

Sin embargo, también se partió de cierta base en el escenario 4, en el que se tuvo que tener muy en cuenta en que cada worker (o en nuestro caso thread de un worker) tendría un 60% de posibilidades de no funcionar correctamente, provocando un gran retraso en el envío de peticiones y resultados. En este caso escogimos hasta 3 threads por cada worker para trabajar con la petición, un número no lo suficientemente alto como para provocar lentitud en la máquina, pero con el que la probabilidad de fallo entre los 3 threads fuese la menor posible.

Para garantizar la posible variabilidad en las mediciones, se han tomado las siguientes medias:

Escenario 1:

Tiempo de ejecución: 1132.4

Tiempo total: 1398.3

Escenario 2:

Tiempo de ejecución: 1942.6

Tiempo total: 1948

Escenario 3:

Tiempo de ejecución: 1238.2

Tiempo total: 1243.5

Escenario 4:

Tiempo de ejecución: 1346.6

Tiempo total: 1473.2

8. Conclusión

A lo largo de esta práctica se ha aprendido a cómo implementar diferentes arquitecturas en el lenguaje de Elixir, en concreto la comunicación síncrona, utilizado en el escenario 1, el servidor concurrente, utilizado en el escenario 2, y la arquitectura master worker, usada en los dos últimos escenarios. Las diferentes arquitecturas fueron elegidas principalmente en base al número de peticiones por segundo por parte del cliente, siendo necesaria una estructura más compleja según estas aumentasen.

A la hora de elegir el número de threads o de workers en las diferentes arquitecturas solucionadas, se utilizó principalmente una estrategia de prueba y error, aumentando o disminuyendo dicho número según fuese necesario.

Una arquitectura software que pudiese dar soporte a los cuatro escenarios simultáneamente sería el master worker, debido principalmente a que , de las arquitecturas utilizadas, solo esta sería capaz de no fallar o que se incumpla el QoS, aunque para los primeros escenarios, esta arquitectura no llegaría a desarrollar todo su potencial.

9. Referencias bibliográficas

Para realizar esta práctica, se ha apoyado en los guiones y documentación proporcionada por los profesores de la asignatura, además de haber seguido diferentes patrones o estilos aprendidos en guías de elixir localizadas en esta página: https://elixirschool.com/en/