Introducción a las arquitecturas de sistemas distribuidos

Eduardo Gimeno Soriano 721615

Sergio Álvarez Peiro 740241

23 de octubre de 2019

Introducción

El sistema consta de tres escenarios diferentes y está formado por dos elementos diferenciados. Un cliente que realiza múltiples peticiones dependiendo del escenario y un servidor que las resuelve y envía la respuesta correspondiente.

La arquitectura del servidor cambia para cada escenario dependiendo de la carga de trabajo que implique cada escenario. Las arquitecturas que se han tenido en cuenta han sido servidor secuencial, servidor concurrente y master-worker.

Sección Principal

Para cada escenario se ha creado un fichero elixir diferente, para el escenario 1 escenario1.exs, para el escenario 2 escenario2.exs y para el escenario 3 escenario3.exs.

Características técnicas de las máquinas utilizadas

Para realizar esta práctica se han utilizado tres máquinas distintas:

- Máquinas lab102-XXX (siendo XXX un número comprendido entre 191 y 210)
    - Núcleos: 4
    - Memoria: 11 GB
    - Disco: 1 TB (500 MB por usuario)
- Máquina ASUS (Eduardo)
    - Núcleos: 4
    - Memoria: 8 GB
    - Disco: 1 TB
- Máquina HP (Sergio)
    - Núcleos: 8
    - Memoria: 8 GB
    - Disco: 128 GB

Las máquinas personal se han utilizado para probar únicamente que el código desarrollado funcionaba bien, para el despliegue del sistema se han utilizado las máquinas de laboratorio 1.02.

Análisis de la carga de trabajo para cada escenario

Vamos a analizar la carga de trabajo que tiene el servidor como el número de peticiones que el cliente realiza por segundo y lo que tarda en atenderlas y enviar una respuesta.

En el primer escenario, la función del cliente que genera workload hace launch del cliente, es decir, una petición y se duerme 2000 milisegundos todo el rato. Esta implementación hace que el número de peticiones sea constante en el tiempo, por lo que la carga de trabajo es constante en el tiempo: 2 peticiones por segundo.

En el segundo escenario la carga de trabajo sigue siendo constante, ya que se siguen haciendo 2 peticiones por segundo, pero el trabajo requerido para atender cada petición es mayor.

En el tercer escenario la carga de trabajo es variable en el tiempo. La función que genera workload para este escenario se comporta de manera distinta. Al principio se hacen peticiones cada 2 segundos y luego a partir de la tercera petición se hacen cada un tiempo aleatorio, pero con el tiempo se hacen más rápidas hasta que la novena vuelven a ser más lentas y se reincia el ciclo.

Patrones arquitecturales para los distintos escenarios

Para resolver los escenarios planteados en la práctica se ha seguido una metodología icremental, es decir, se ha comenzado desde la solución más sencilla y a partir de ahi se ha ido evolucionando según las necesidades de cada escenario.

Partiendo de lo anterior, para resolver el primer escenario se ha comenzado probando con una arquitectura cliente-servidor secuencial. Para comprobar que era la solución idónea se ha comprobado para cada petición del cliente que la QoS no era violada, en la sección Validación Experimental se informa más al detalle de la comprobación de cada solución. Para este primer escenario esta solución ha sido suficiente, ya que la QoS se respetaba en cada petición del cliente.

Para el segundo escenario se ha partido del escenario anterior, pero la solución cliente-servidor secuencial para este escenario no cumplía la QoS, por tanto se ha pasado a la siguiente arquitectura, cliente-servidor concurrente. Esta solución si ha sido suficente para este escenario, respetando la QoS para cada petición del cliente.

Para el úñtimo escenario se ha seguido el mismo procedimiento, se ha partido de la solución del escenario 2 y se ha comprobado que la QoS no se respetaba, por tanto se ha pasado a la siguiente arquitectura, master-worker. En una primera instancia se ha probado con un único worker. Como esta arquitectura presenta dos posibles soluciones, se ha optado por establecer el número de workers necesarios para garantizar la QoS en el momento de mayor carga de trabajo. En primer lugar se ha probado con un único worker, pero no ha sido suficiente para soportar dicho momento, por ello, se ha añadido un worker más y en esta ocasión, la QoS si era respetada para petición del cliente. Añadir que la implementación fibonacci_tr siempre da problemas con la QoS, ya que esta está establecida para la implementación fibonacci.

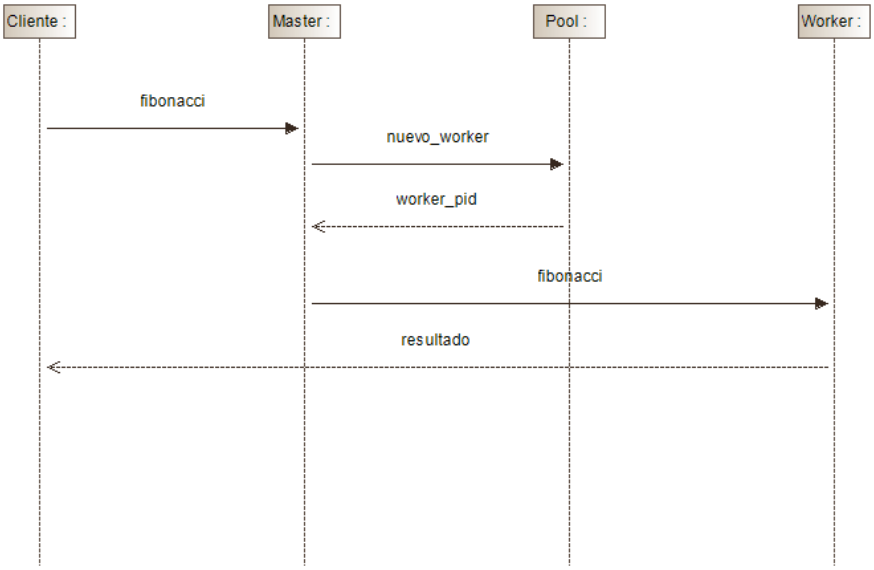

Figura 1: D.Secuencia escenario 3

Arquitectura software para cubrir los tres escenarios simultáneamente

La arquitectura necesaria para cubrir los tres casos simultáneamente sería la utilizada para el escenario 3, porque este escenario es el que genera mayor carga de trabajo para el servidor, por tanto, si se consigue garantizar la QoS para este escenario, para los dos anteriores también queda garantizada. El único aspecto que podría variar sería el número de workers necesario, ya que se pasaría a tener tres clientes realizando peticiones en vez de uno, siguiendo la solución realizada para el escenario 3, habría que establecer el número de workers necesario para soportar el momento con mayor carga de trabajo.

Validación Experimental

Para validar la solución de cada escenario se han utilizado las máquinas de laboratorio 1.02. Se han tomado 10 mediciones de la ejecución con la implemetnación fibonacci del tiempo necesario para resolver la operación a realizar y el tiempo total que conlleva la petición. Para la implementación fibonacci_tr se han tomado también algunas medidas (no consecutivas), pero teniendo más relevancia la implementación anterior.

Escenario 1				
Implementación	T. Aislada	Media (A)	T. Total	Media (T)
fibonacci	1499	1507,6	1499	1509,1
fibonacci	1409		1411	
fibonacci	1508		1510	
fibonacci	1523		1524	
fibonacci	1522		1524	
fibonacci	1524		1526	
fibonacci	1524		1524	
fibonacci	1523		1525	
fibonacci	1520		1522	
fibonacci	1524		1526	
Escenario 2				
fibonacci	1555	1531	1557	1532,3
fibonacci	1557		1558	
fibonacci	1558		1559	
fibonacci	1492		1494	
fibonacci	1493		1494	
fibonacci	1535		1537	
fibonacci	1536		1537	
fibonacci	1538		1539	
fibonacci	1554		1555	
fibonacci	1492		1493	
Escenario 3				
fibonacci	1564	1587,4	1566	1589,3
fibonacci	1566		1568	
fibonacci	1567		1569	
fibonacci	1569		1571	
fibonacci	1598		1600	
fibonacci	1599		1601	
fibonacci	1603		1605	
fibonacci	1601		1602	
fibonacci	1600		1602	
fibonacci	1607		1609	
fibonacci_tr	0	0	2	2,4
fibonacci_tr	0		3	
fibonacci_tr	0		2	
fibonacci_tr	0		3	
fibonacci_tr	0		2	

Figura 2: Tabla de mediciones

Como se puede observar, en la implementación de fibonacci_tr no llega ni siquiera a un milisegundo la tarea aislada, por lo que no es posible que la QoS dada en el enuciado se pueda a llegar a cumplir.

Conclusiones

A la hora de buscar una arquitectura para un sistema distribuido es necesario buscar la solución más simple que satisfaga los requisitos de dicho sistema para no dedicar más recursos de los necesarios. Si esta solución no es suficente, incrementar dicha solución paso a paso hasta poder resolver el problema.

La QoS es un contrato que se firma y se debe cumplir siempre para garantizar el correcto funcionamiento del sistema.

Bibliografía

[1] Material de la asignatura de Sistemas Distribuidos del grado de Ingeniería Informática de UNIZAR.

[2] Documentación del lenguaje Elixir https://elixir-lang.org/docs.html