Práctica 1. Introducción a las arquitecturas de Sistemas Distribuidos

Andrés Gavín Murillo 716358

Borja Aguado 741440

25 de octubre de 2018

Introducción

El sistema, que consta de cuatro escenarios diferentes, incluye un cliente que realiza múltiples peticiones (dependiendo del escenario) y un servidor que las resuelve.

La arquitectura software del servidor varía según los requisitos funcionales de cada escenario, es decir, dependiendo de la carga de trabajo en cada escenario, se resolverá de manera secuencial, concurrente o distribuida.

Sección Principal

Para cada tipo de escenario existe un fichero elixir distinto (*escenario1.exs*, *escenario2.exs*, *escenario3.exs* y *escenario4.exs*).

Arquitectura software de los escenarios

En el escenario 1, el cliente lanza una petición cada 2 segundos. La petición es resuelta en, aproximadamente, 1 segundo de tiempo de ejecución. Esto ha sido resuelto con la implementación de un servidor secuencial.

En el escenario 2, el cliente lanza N peticiones [1] cada 2 segundos. Cada petición es resuelta en, aproximadamente, 1 segundo de tiempo de ejecución. Esto ha sido resuelto con la implementación de un servidor concurrente.

En el escenario 3, el cliente lanza N * 2 + 2 peticiones cada 2 segundos. Cada petición es resuelta en, aproximadamente, 1 segundo de tiempo de ejecución. Esto ha sido resuelto con la implementación de un servidor distribuido *master-worker*.

En el escenario 4, el cliente lanza N * 2 + 2 peticiones cada 2 segundos. Cada petición es resuelta en, aproximadamente, entre 1 y 3 segundos de tiempo de ejecución. Esto ha sido resuelto con la implementación de un servidor distribuido *master-worker*.

[1] N equivale al número de *cores* que tiene la máquina donde se ejecuta el servidor.

Implementación de los servidores

En el escenario 1, el servidor es secuencial al mantener el código fuente proporcionado en la práctica. Es decir, cada vez que le llega una petición del cliente la resuelve.

En el escenario 2, el servidor es concurrente al generar un nuevo *thread* en su ejecución para resolver cada petición del cliente. Esto se consigue mediante la utilización de la función *spawn()*.

En los escenarios 3 y 4, el servidor es distribuido al generar un nuevo *thread* en uno de sus *workers* para resolver cada petición del cliente de manera equitativa y cíclica, es decir, va distribuyendo la carga de trabajo en los *workers* ordenadamente. Esto se consigue mediante la utilización de la función *Node.spawn()*.

Ejecución del sistema

En el escenario 1 y 2 la puesta en ejecución del sistema se realiza como sigue:

1. **S**: iex --name nodo1@*IP_nodo1* --cookie ssdd
2. **C**: iex --name nodo2@*IP_nodo2* --cookie ssdd
3. **S**: Process.register(self(), :server)
4. **C**: Node.connect(:"nodo1@*IP_nodo1* ")
5. **S**: Perfectos.servidor()
6. **C**: (escenario 1): Perfectos_cliente.cliente({:server, :"nodo1@*IP_nodo1*"}, :uno)
7. **C**: (escenario 2): Perfectos_cliente.cliente({:server, :"nodo1@*IP_nodo1*"}, :dos)

En el escenario 3 y 4 la puesta en ejecución del sistema se realiza como sigue:

1. **S**: iex --name nodo1@*IP_nodo1* --cookie ssdd
2. **C**: iex --name nodo2@*IP_nodo2* --cookie ssdd
3. **W**: iex --name nodoN@*IP_nodoN* --cookie ssdd
4. **S**: Process.register(self(), :server)
5. **C**: Node.connect(:"nodo1@*IP_nodo1* ")
6. **W**: Node.connect(:"nodo1@*IP_nodo1* ")
7. **S**: Perfectos.master()
8. **C**: (escenario 3): Perfectos_cliente.cliente({:server, :"nodo1@*IP_nodo1*"}, :tres)
9. **C**: (escen. 4): Perfectos_cliente.cliente({:server, :"nodo1@*IP_nodo1*"}, :cuatro)

[**S**]: Máquina donde se ejecuta el servidor.

[**C**]: Máquina donde se ejecuta el cliente.

[**W**]: Máquina(s) donde se ejecuta(n) el(los) *worker*(s).

Validación

Para esta validación se utilizan los ordenadores del laboratorio que tienen 4 *cores* cada uno. El cliente es ejecutado en una máquina diferente por simplicidad.

En el escenario 1, el servidor se compone de 1 máquina, emplea 1 *core* y resuelve hasta 2 peticiones cada 2 segundos, frente a 1 petición cada 2 segundos que lanza el cliente.

En el escenario 2, el servidor se compone de 1 máquina, emplea hasta 4 *cores* y resuelve hasta 8 peticiones cada 2 segundos, frente a 4 peticiones cada 2 segundos que lanza el cliente.

En el escenario 3, el servidor se compone de 2 máquinas, 1 *core* para el *master* y 7 *cores* para los *workers,* y resuelve hasta 14 peticiones cada 2 segundos, frente a 10 peticiones cada 2 segundos que lanza el cliente.

En el escenario 4, el servidor se compone de 4 máquinas, 1 *core* para el *master* y 15 *cores* para los *workers,* y resuelve hasta 15 peticiones cada 3 segundos, es decir, 30 peticiones cada 6 segundos, frente a 10 peticiones cada 2 segundos que lanza el cliente, es decir, 30 peticiones cada 6 segundos.

Conclusiones

Las arquitecturas en los sistemas han de satisfacer los requisitos funcionales de estos, sin exceder las prestaciones. Es decir, si un sistema requiere 1 *core* para funcionar, no tiene sentido dedicarle más *cores*.

En los escenarios presentes, el escenario 4 resuelve todos ellos simultáneamente, aunque para ello se debieran de conectar más máquinas (y emplear más recursos) que si sólo se resuelve uno de los escenarios en concreto.

Bibliografía

[1] Material de la asignatura Sistemas Distribuidos en Ingeniería Informática de la Universidad de Zaragoza durante el curso 2018/19.

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