Memoria técnica

Sistemas distribuidos – Práctica 1

Grado de Ingeniería Informática, 2019-2020
Universidad de Zaragoza

Alberto Calvo Rubió
760739

Óscar Baselga Lahoz
760077

Índice

Introducción

En esta práctica se analizan las arquitecturas cliente-servidor y master-worker. Se implementan 3 escenarios diferentes en los que se realizan pruebas para evaluar si cumplen con un QoS establecido.
Además, al master-worker se añade un nuevo componente Pool, con la importante tarea de gestionar los Workers, que para ser incluido en el sistema van a tomarse importantes decisiones de diseño.

Cuestiones

Características de las máquinas del laboratorio 1.02

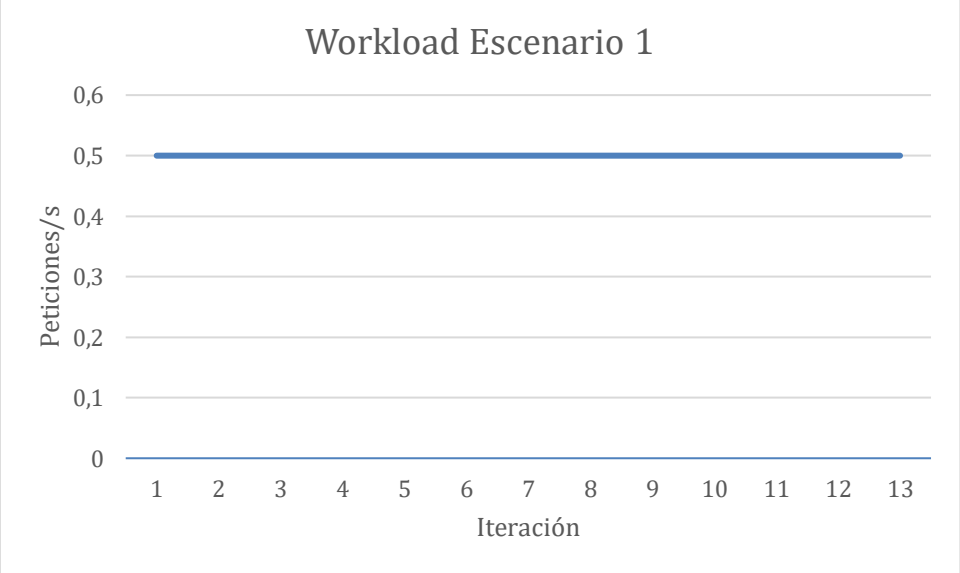
Se tratan de máquinas con un procesador de 4 núcleos (lo que significa que como máximo se pueden ejecutar 4 Workers al mismo tiempo), una memoria de 11.6 GiB y un disco de 49.5 GB.

Carga de trabajo de cada escenario

Escenario 1

En este escenario solo encontramos un solo caso en el que se realiza una petición cada 2 segundos.

Peticiones/Segundo = 0.5

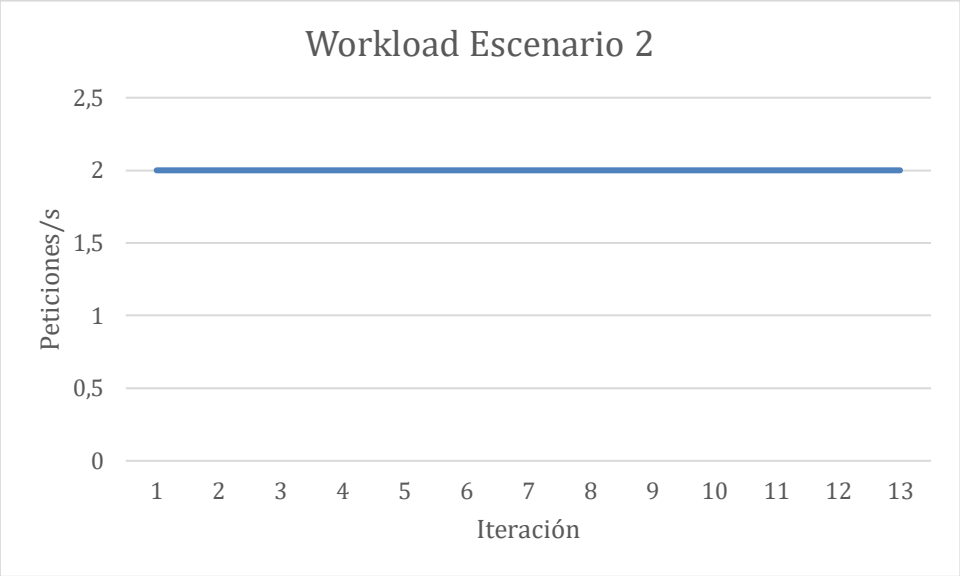


Se trata de una carga de trabajo uniforme en todas las iteraciones

Escenario 2

Igual que en el primer escenario solo existe un caso, se realizan 4 peticiones en 2 segundos.

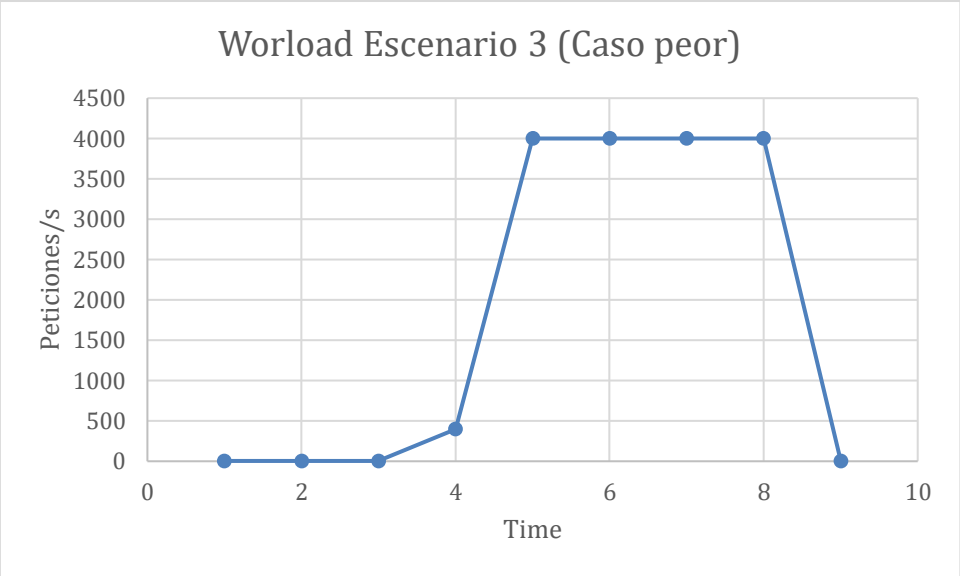
Peticiones/Segundo = 2



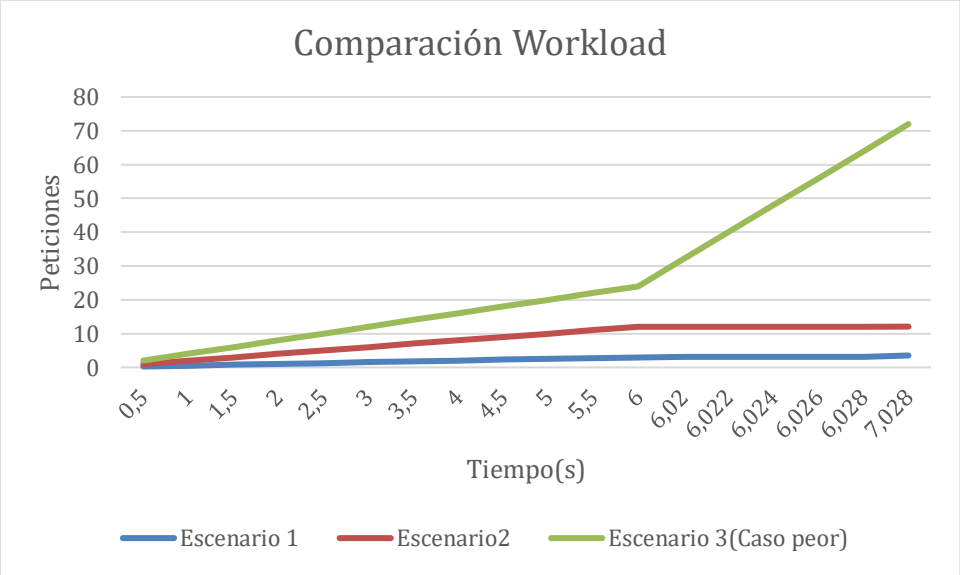
Escenario 3

Este escenario tiene diferentes configuraciones de peticiones a lo largo de su ejecución. Al comienzo, cuando la variable time vale de 1 a 3, se mantienen las peticiones por segundo. En cambio, al llegar a 4 y superiores, tenemos un factor aleatorio que puede variar el tiempo de espera para enviar las peticiones. Finalmente, cuando su valor es 9 se recupera valores menos extremos. A continuación, se van a exponer los casos mejores y peores en cada iteración.

		Tiempo(s)		Peticiones/Tiempo(s)	
Time	Peticiones	Caso mejor	Caso peor	Caso mejor	Caso peor
1-3	8	2		4	
4	8	2	0,02	4	400
5-8	8	0,02	0,002	400	4000
9	8	2	1	4	8



Comparación escenarios



En esta gráfica se muestra como evoluciona el número de peticiones dependiendo del escenario. Es apreciable como en el escenario 3 aumentan rápidamente conforme avanza el tiempo en comparación con los otros escenarios, por lo que serán necesarios más recursos para soportar tal carga de trabajo.

Tiempos de ejecución

La carga de trabajo viene definida por el cálculo de la función de Fibonacci sobre cada elemento de una lista (en este caso 1..36). Las dos funciones a estudiar son: Fib.fibonacci(), la cual cuenta con un tiempo de ejecución medio de 1488 milisegundos, y Fib.fibonacci_tr(), cuyo tiempo de ejecución es aproximadamente nulo.

Calidad de servicio (QoS)

La condición que se ha impuesto en esta práctica para el QoS consiste en que el tiempo total de respuesta sea menor que 1.5 veces el tiempo de ejecución de la tarea de forma aislada.

La fórmula que calcula el tiempo total es:

$$TiempoTotal = Ttransmision + Tejecucion + Tespera$$

El tiempo total se ha medido guardando el valor inicial del tiempo (Time.utc_now()) y pasándola a un nuevo proceso que se encarga de esperar la respuesta por parte del Servidor (cliente-servidor) o Worker (master-worker). Una vez que llega, se realiza la diferencia con el tiempo actual.

El tiempo de ejecución se ha medido de forma similar a la anterior pero el tiempo inicial se guarda antes de ejecutar Enum.map que calcula la función Fib.fibonacci() en toda la lista. Y el tiempo final justo cuando termina.

Modificaciones en el Cliente

Para evitar que se atiendan las peticiones secuencialmente, se ha creado un proceso escuchar() que se encarga de recibir el resultado y mostrar el contador del tiempo total.

Patrones arquitecturales

En cada uno de los escenarios se puede utilizar cualquiera de las dos arquitecturas, sin embargo, no en todas se cumple el QoS. Por ello, se ha elegido la que más se adapta a cada uno.

Escenario 1

Se ha implementado un cliente-servidor ya que, la carga de trabajo (peticiones) se envían de forma secuencial, es decir, se envía la petición y no se sigue hasta haber recibido el resultado de la anterior.
Este tipo de arquitectura es la más sencilla que cumple con el QoS.
Por otro lado, la utilización del master-worker seguiría cumpliendo con el QoS aunque su implementación es más compleja.

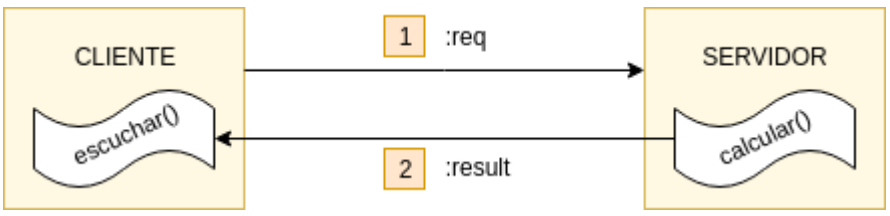


Diagrama cliente-servidor

Escenario 2

En este caso, también se utiliza un cliente-servidor, pues se sigue manteniendo el QoS ya que la carga de trabajo solo es de 4 peticiones por cada 2 segundos, pudiendo el servidor hacerse cargo de todos ellos.

De igual forma, puede implementarse con un master-worker aunque no es necesario.

Escenario 3

Ahora resulta útil implementarlo como master-worker debido a que aumenta en gran medida la carga de trabajo y no puede ser soportada por un sólo servidor, siendo necesarias más máquinas. Con esta arquitectura, se pueden añadir más máquinas como servidores donde se ejecutan los cálculos, repartiendo la carga de trabajo para cumplir el QoS. En el siguiente apartado se explica con más detalle.

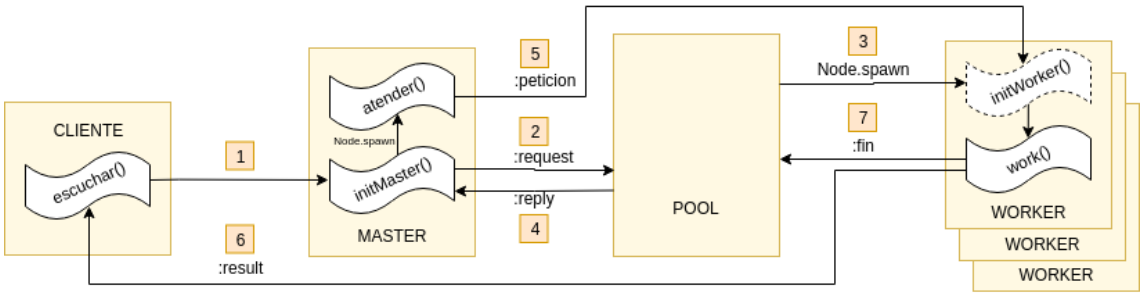
A la hora de la práctica, han sido necesarias 5 máquinas en las que poder crear workers para mantener el QoS exigido.

Arquitectura software con soporte a los tres escenarios

Master-Pool-Worker

El sistema consta de 4 partes diferenciadas: Cliente, Master, Pool y Worker.
El funcionamiento del sistema consiste en:

1. El Cliente manda una petición al Master donde detalla el problema que quiere solucionar (en este caso Fibonacci).
2. El Master lanza un proceso (atender()), que va a ser específico para cada cliente. Este proceso manda una petición al Pool exigiendo un proceso Worker que pueda solucionar el problema del Cliente.
3. Una vez reciba el Pool la petición, va a consultar las cargas de trabajo de todas las máquinas Worker y escoge la que menos tenga. Si esta carga es menor que el número de núcleos de la máquina, el Pool lanzará un proceso nuevo en dicha máquina y su PID se mandará al proceso atender() del Master.
4. El proceso atender() envía el problema al proceso Worker que tiene asignado
5. El Worker ejecuta la operación solicitada y envía su resultado al Cliente. Además, envía un mensaje de liberación al Pool. Este va acompañado del nombre del nodo de la máquina Worker para que así el Pool pueda actualizar las cargas de trabajo.



En el caso de que llegue la petición del proceso atender() al Pool y todas las máquinas Worker tenga una carga de trabajo igual al número de núcleos de su procesador, el proceso atender()

se bloqueará esperando la confirmación del Pool, la cual será enviada cuando llegue un nuevo mensaje de liberación al Pool.

Finalmente, cabe destacar la continua creación y destrucción de procesos tanto en la máquina Master como en los Workers. Esta gestión de los procesos en las máquinas Workers, que hace que sean dinámicos, se realiza en función de la carga de trabajo de las máquinas. Es una funcionalidad añadida del sistema que permite distribuir mejor la carga de trabajo. Además, al ser creación dinámica, hace que el sistema sea más flexible en cuanto al número de Workers, ya que para añadir una nueva máquina simplemente hay que incluirla a la lista que se le pasa a Pool.

Bibliografía

Guía de sintaxis:
https://github.com/christopheradams/elixir_style_guide/blob/master/README.md

Referencia Elixir:
https://hexdocs.pm/elixir/Kernel.html