SISTEMAS DISTRIBUIDOS

PRACTICA 1:

Introducción a las Arquitecturas de Sistemas Distribuidos

Alejandro Terron 761069
Óscar Anadón 760628

14-octubre-2021

1. Introducción

En esta memoria se va a abordar el desarrollo de cuatro arquitecturas de carácter cliente-servidor. Una de ellas es secuencial y las otras tres son soluciones distribuidas.

El cliente que se nos ha facilitado genera una carga de trabajo de seis peticiones a intervalos regulares. El servidor a partir de esas peticiones tiene que calcular el número de primos comprendido entre el intervalo proporcionado por el cliente, el cuello de botella principal para desempeñar esta tarea es la CPU.

Para el desarrollo de esta tarea se ha dispuesto del laboratorio L1.02, donde se cuenta con 20 máquinas Little Endian con 6 cores y 32 GB de RAM. A partir de estas especificaciones, el trabajo a realizar consiste en el cálculo teórico y posteriormente empírico de la carga de trabajo máxima sin que se viole el QoS en cada una de las 4 arquitecturas implementadas.

2. Diseño de las 4 arquitecturas

Cliente-servidor secuencial

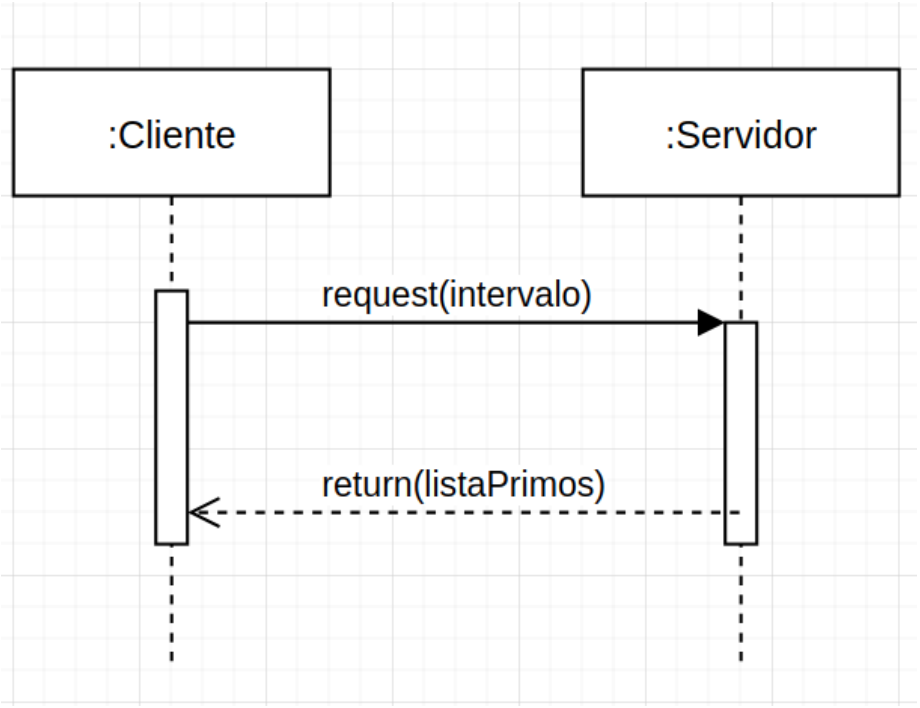

Figura 1: Diagrama de secuencia de la arquitectura secuencial

Cliente-servidor concurrente infinito

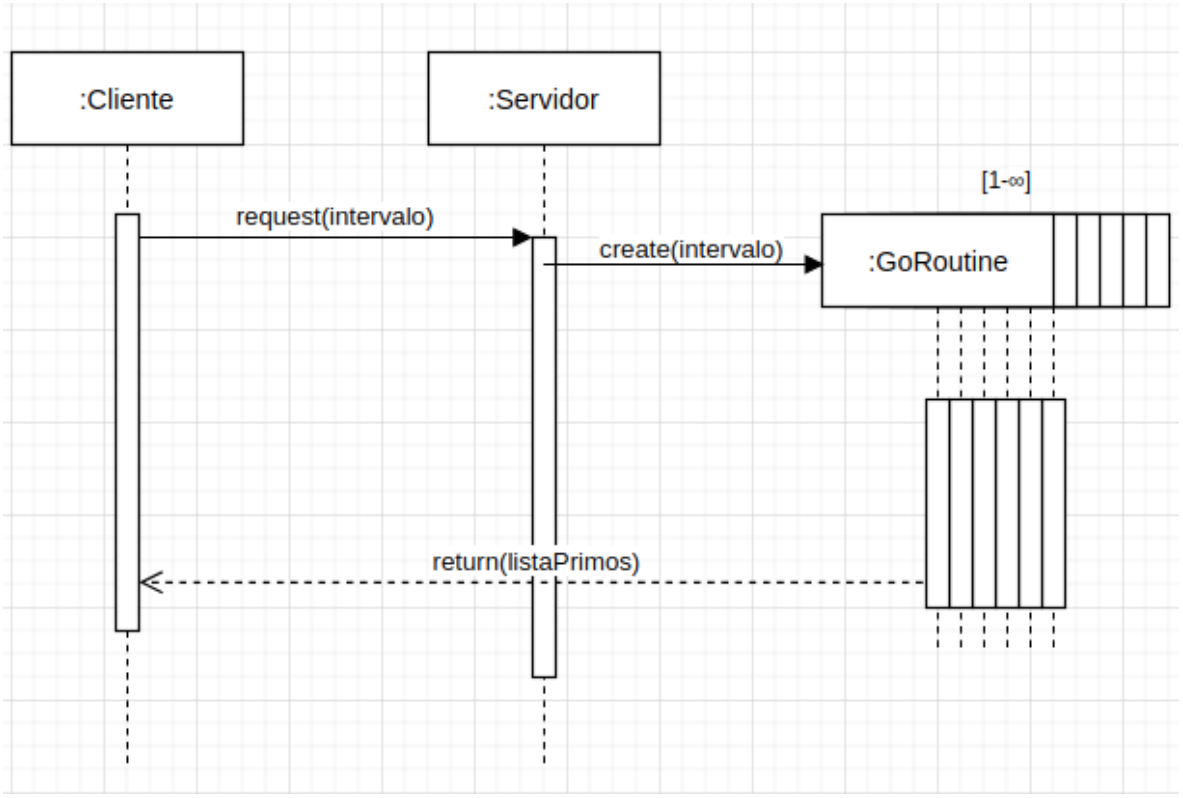

Figura 2: Diagrama de secuencia de la arquitectura distribuida con GoRoutine por petición

Cliente-servidor concurrente con piscina

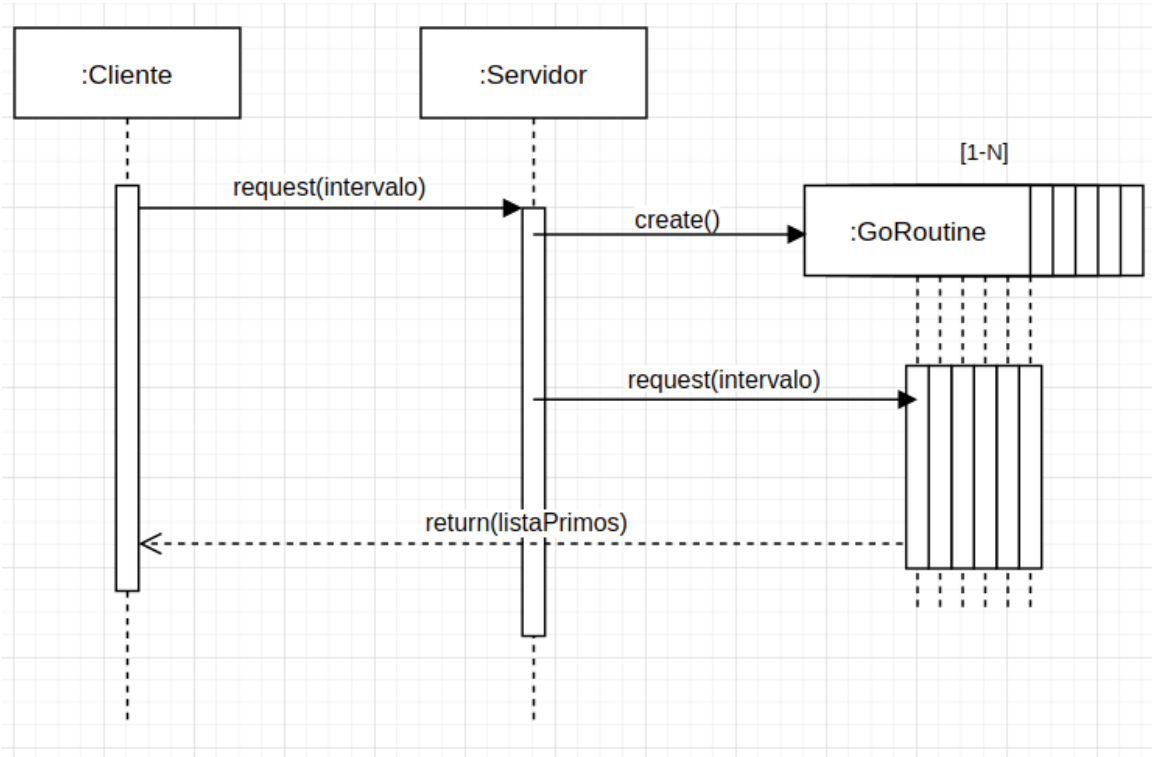

Figura 3: Diagrama de secuencia de la arquitectura distribuida con pool de GoRoutines

Master-Worker

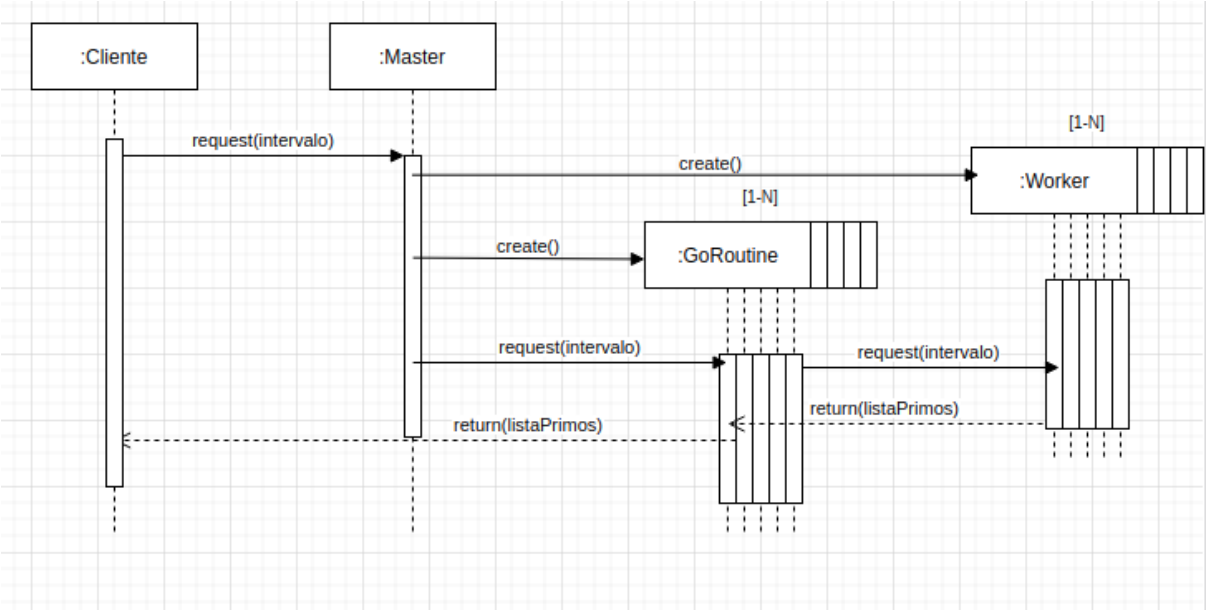

Figura 4: Diagrama de secuencia de la arquitectura Master-Worker

3. Análisis teórico

Para realizar este apartado es necesario conocer el Tex de FindPrimes, para ello se realizaron mediciones aisladas de dicha función obteniendo un tiempo medio de 1.6s por lo que el QoS es de 3.2 segundos.

Cliente-servidor secuencial

Se procesan las peticiones una detrás de otra, por ello la máxima carga de trabajo que puede soportar es la de una tarea por tiempo de ejecución. En cuanto el sistema tenga que soportar una carga de trabajo mayor a la anteriormente descrita se generará una cola de tareas pendientes.

Carga máx=1/1.6=0.62 tareas segundo

Cliente-servidor concurrente infinito

Se procesan las peticiones conforme llegan sin ninguna restricción, por ello la máxima carga de trabajo que puede soportar está determinado por la CPU en la que se ejecute, en este caso se disponen de 6 cores por lo tanto 6 tareas por tiempo de ejecución.

Carga máx=6/1.6=3.75 tareas segundo

Cliente-servidor concurrente con piscina

Se procesan las peticiones conforme llegan hasta un límite estipulado previa ejecución, en este caso se disponen de 6 GoRoutines como pool para exprimir al máximo los recursos computacionales, por lo que como en la anterior lo óptimo son 6 tareas por tiempo de ejecución.

Carga máx=6/1.6=3.75 tareas segundo

Master-Worker

Se procesan las peticiones conforme llegan hasta un límite estipulado previa ejecución, a groso modo es una extensión de la anterior arquitectura anterior solo que en vez de utilizar los recursos del ordenador donde se ejecuta el servidor (Master), se hace uso de otros por lo que la escalabilidad de esta arquitectura es superior a todas las anteriores.

Carga máx=min(Worker,GoRoutines que sirven a estos)/1.6

4. Análisis práctico

Se han ejecutado dos test por arquitectura, uno dentro de los márgenes de cumplimiento del QoS y otro fuera de ellos, el valor escogido para violar el QoS ha sido seleccionado a partir de las tareas por minuto * 2 y se ha aproximado a la cota superior.

Cliente-servidor secuencial

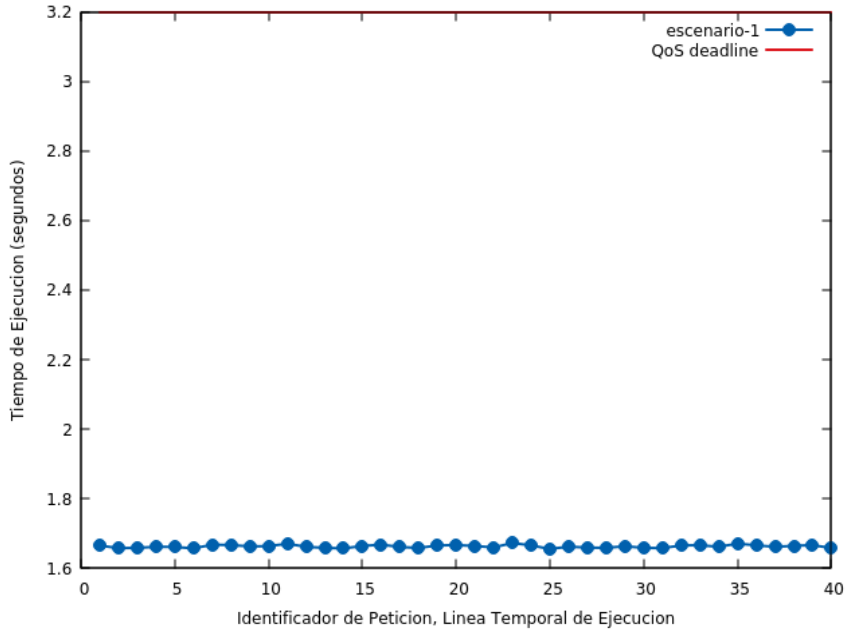

Figura 5: Test secuencial con una petición cada 2 segundos para asegurar QoS

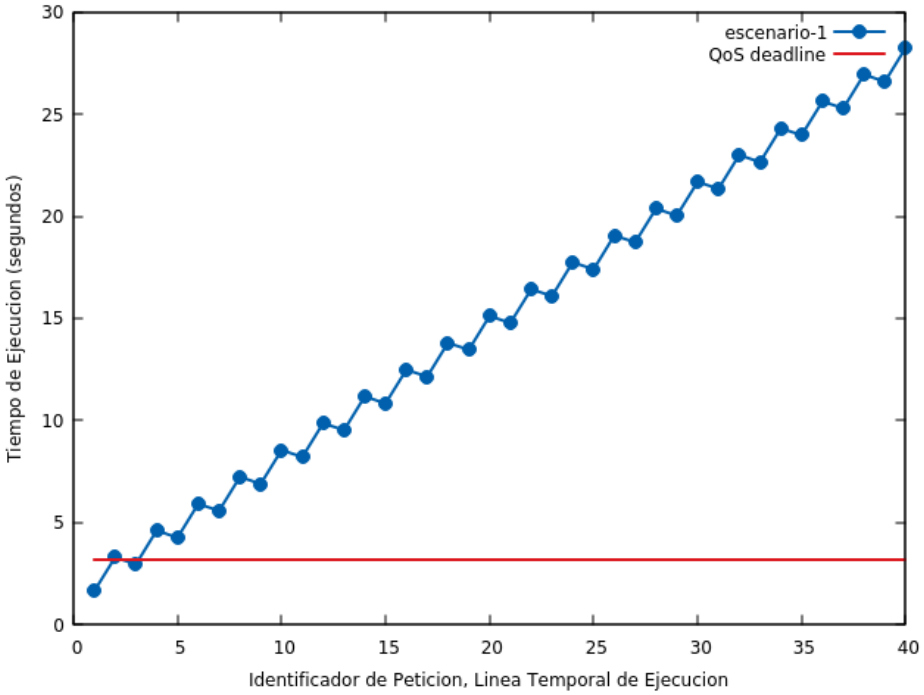

Figura 6: Test secuencial con 2 peticiones cada 2 segundos para violar QoS

Cliente-servidor concurrente infinito

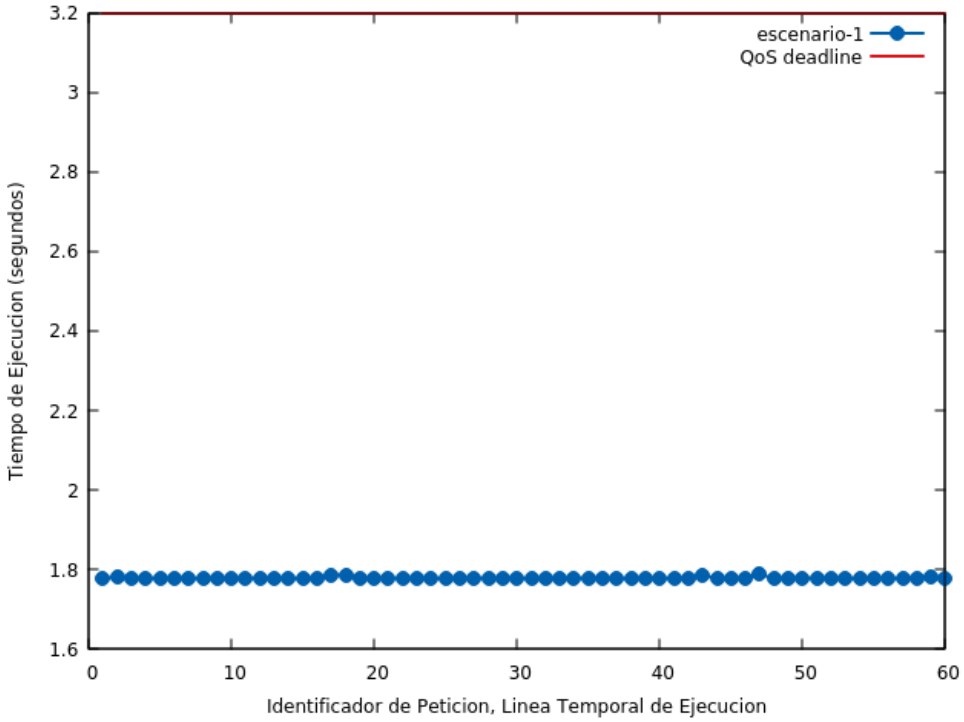

Figura 7: Test GoRoutine petición con 6 peticiones cada 2 segundos para asegurar QoS

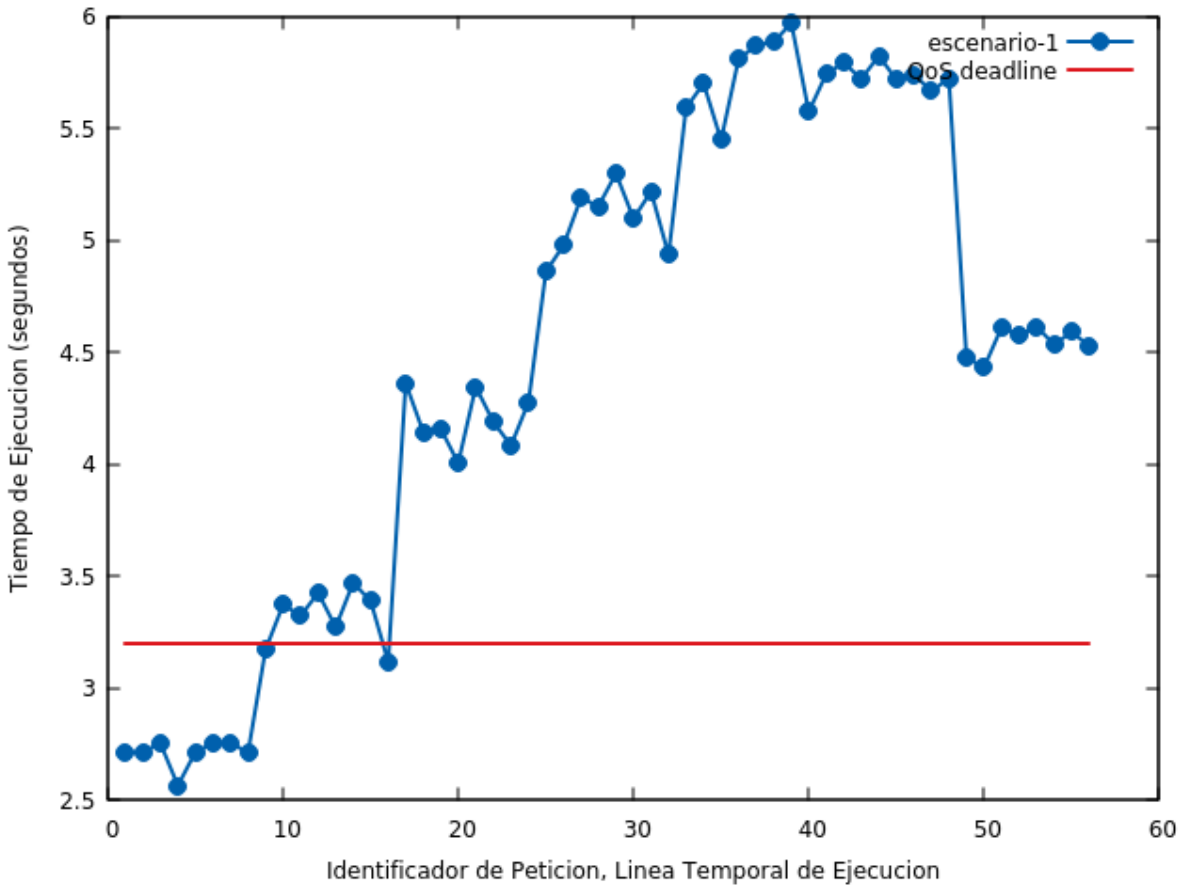

Figura 8: Test secuencial con 8 peticiones cada 2 segundos para violar QoS

Cliente-servidor concurrente con piscina

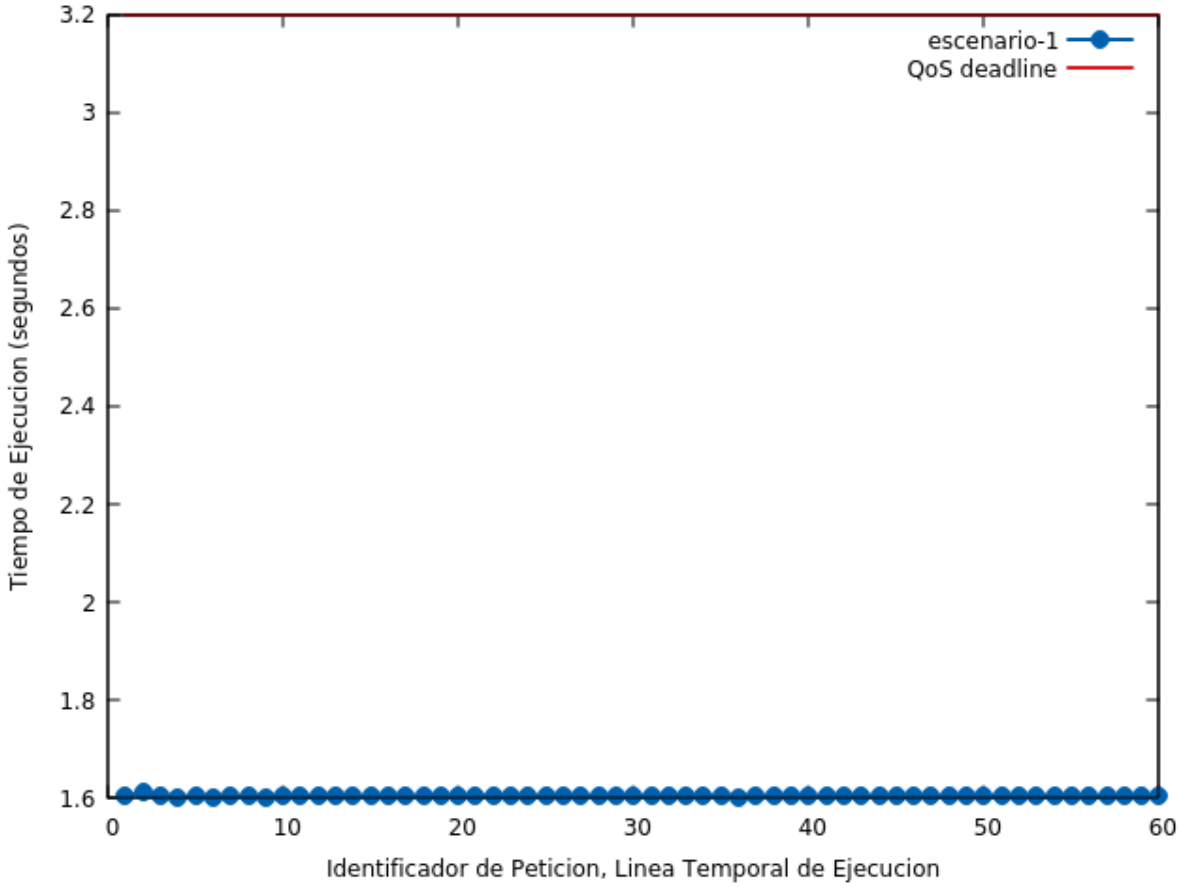

Figura9: Test pool de 6 con 6 peticiones cada 2 segundos para asegurar QoS

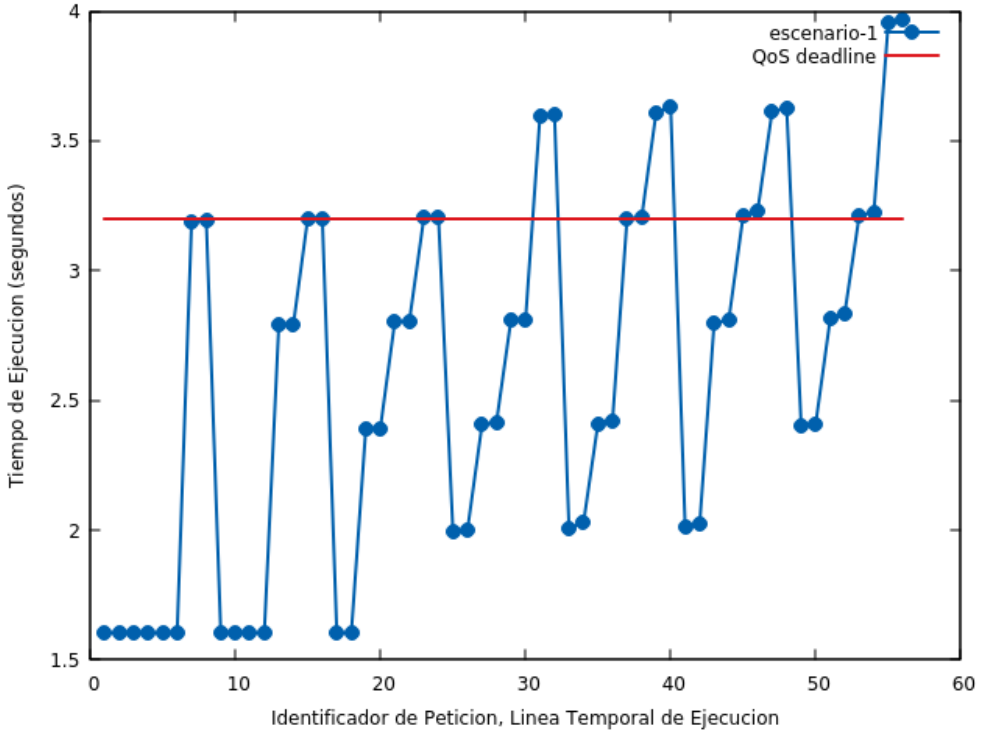

Figura10: Test pool de 6 con 8 peticiones cada 2 segundos para violar QoS

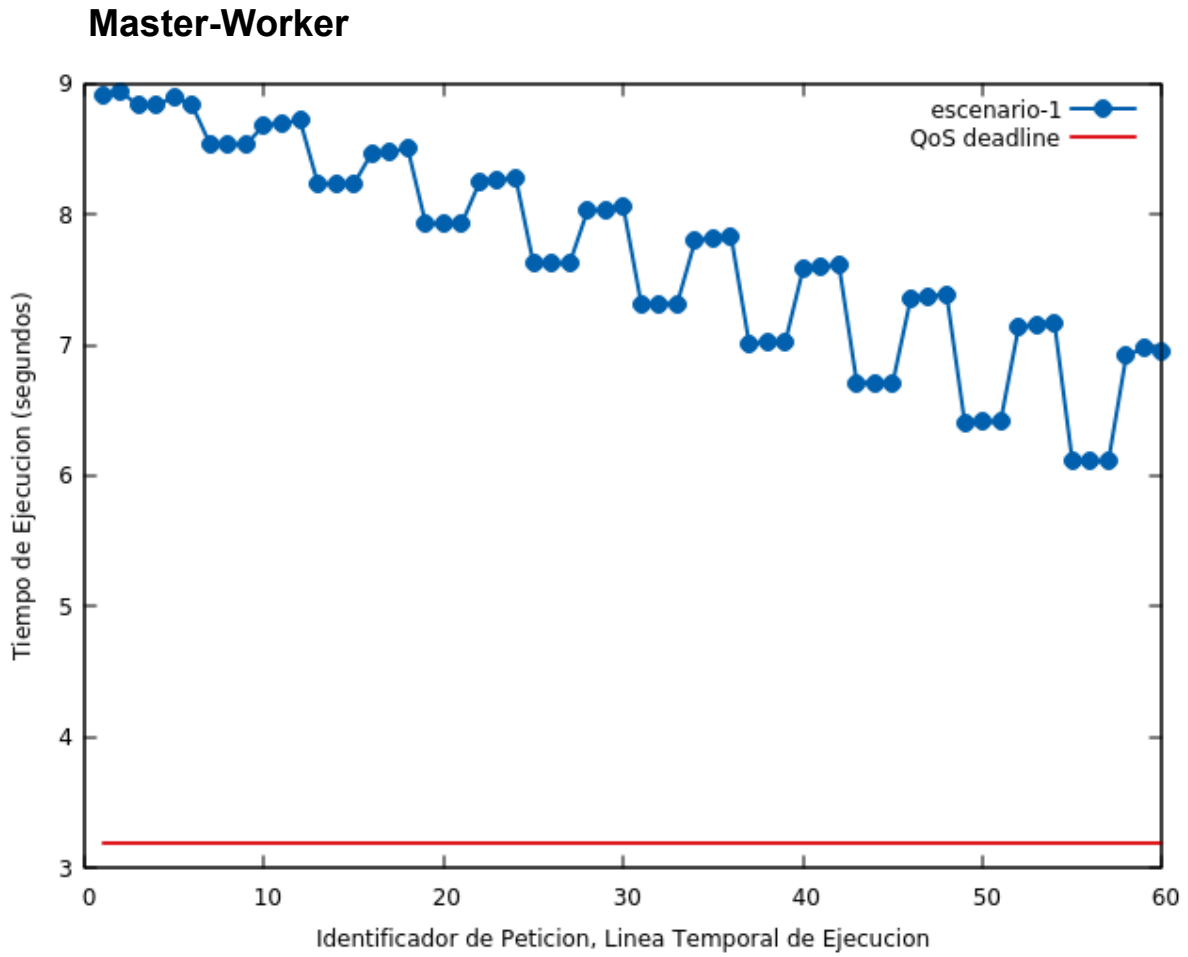

Figura11: Test MasterWorker de 6 workers con 6 peticiones cada 2 segundos para asegurar QoS

No se han seguido realizando pruebas porque en este caso en concreto se viola el QoS cuando no tendría porque, en la prueba el cliente, el servidor y los workers 3 a 3 se ejecutan en máquinas separadas. Esto es debido a que hay dormir al proceso Master para darle tiempo a los workers a hacer deploy, de todas maneras como se puede ver al estar teoricamente por debajo de imcumplir el QoS la gráfica presenta un comportamiento descendente.