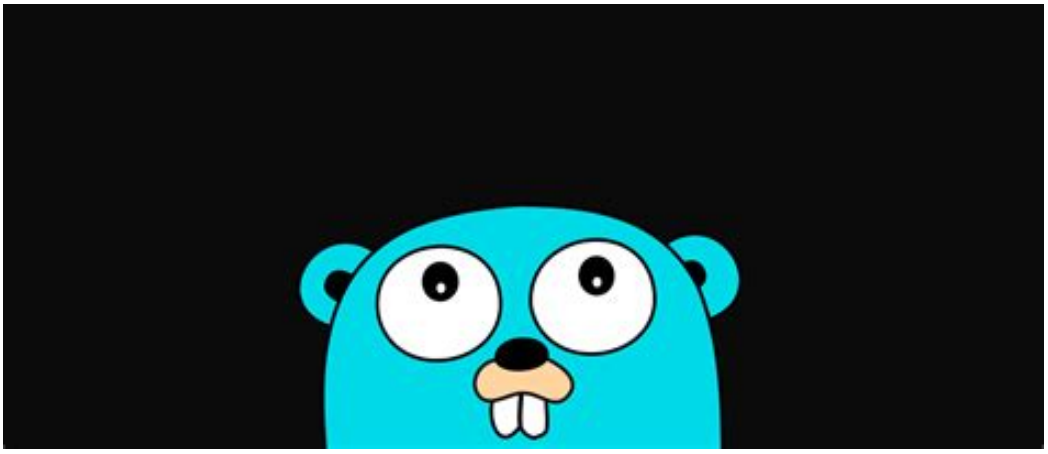
Práctica 1: Introducción a Sistemas Distribuidos

Sistemas Distribuidos
Universidad de Zaragoza, curso 2022/2023



Ayelen Nuño Gracia 799301
Loreto Matinero Angusto 796598

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Descripción del problema:

Se va a diseñar e implementar 4 arquitecturas usando diversas metodologías para permitir la ejecución de una aplicación distribuida de cálculo de primos en un intervalo determinado. Para ello se va a utilizar el lenguaje go. La comunicación del cliente con el servidor se realizará utilizando el protocolo TCP, adicionalmente se utilizarán canales para la comunicación entre los diversos hilos existentes (GoRutinas).

Estas cuatro arquitecturas serán las siguientes:

- La primera de ellas será una ejecución secuencial en la cual se recibirá la petición de los clientes y se atenderán de manera secuencial.
- La segunda consiste en que las peticiones recibidas de los clientes serán gestionadas por goRutinas, creando una goRutina para cada una de las peticiones asignadas por los clientes.
- En tercer lugar se utilizara un pool fijo de Goroutines, es decir se creará un número fijo de goRutinas a las cuales se le asignarán las peticiones de los clientes a través de canales, esto evita tener que estar constantemente creando goRutinas, lo cual permite acelerar la ejecución del proceso ya que la creación de cada una de ellas consume mucho tiempo.
- Por último se seguirá la estructura master-worker en la que se resolverán las peticiones en sistemas externos al nuestro, permitiendo acelerar el proceso de resolución de las peticiones. Para esto se debe tener en cuenta que el master actuará como servidor para los clientes, pero a su vez será quien envíe las peticiones a los workers.

Recursos computacionales utilizados para la resolución de los diversos apartados:

Como se ha mencionado anteriormente se va a utilizar el lenguaje de Go, y la herramienta de VSCodium, de igual forma se hará uso de la herramienta MobaXTerm para conectarse remotamente a los labs.
Por último, para la realización de esta práctica, se ha hecho uso de las máquinas del laboratorio L1.02, las cuales constan de 6 cores de 64 bits.

Diseño (máquinas de estado, diagramas de secuencia)

★ cliente-servidor secuencial

Mapa de estados:

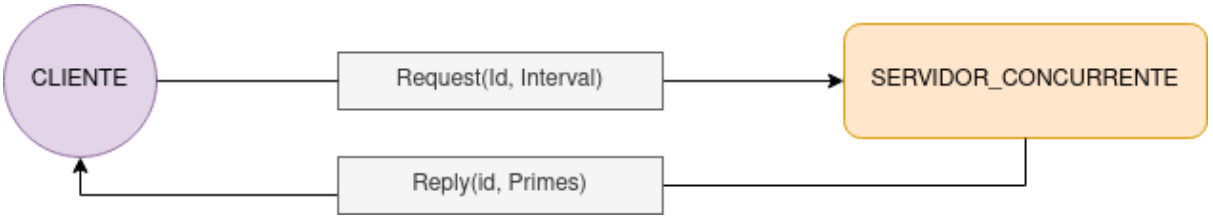
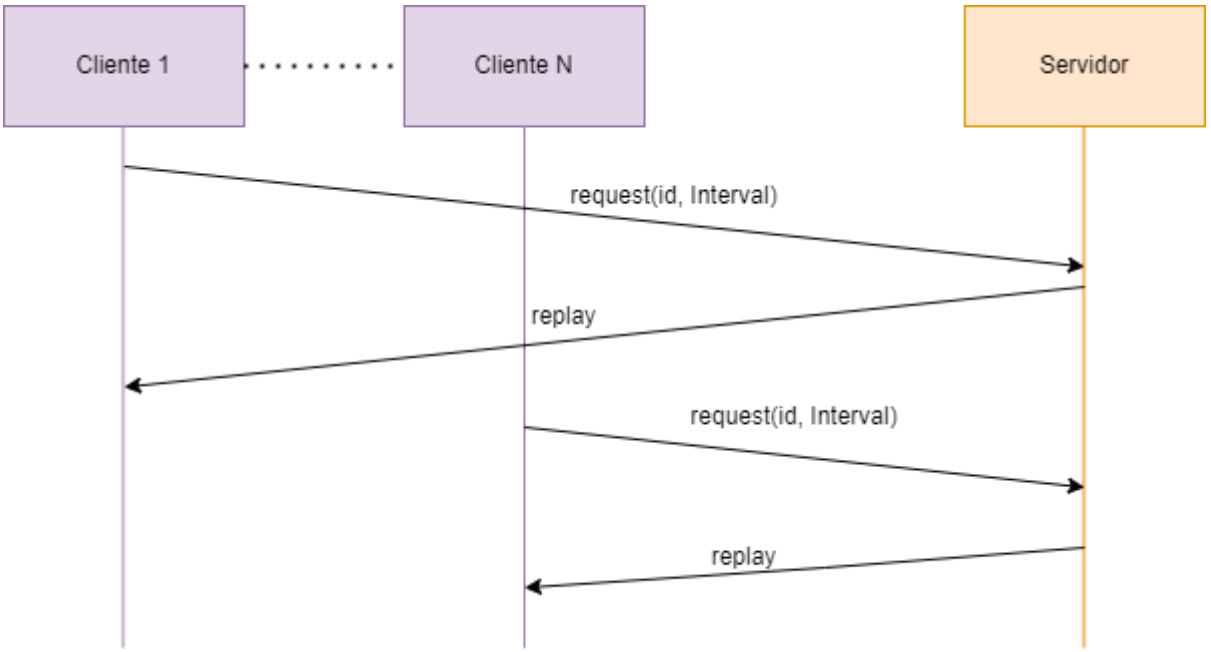


Diagrama de secuencia:



★ cliente servidor concurrente

Mapa de estados:

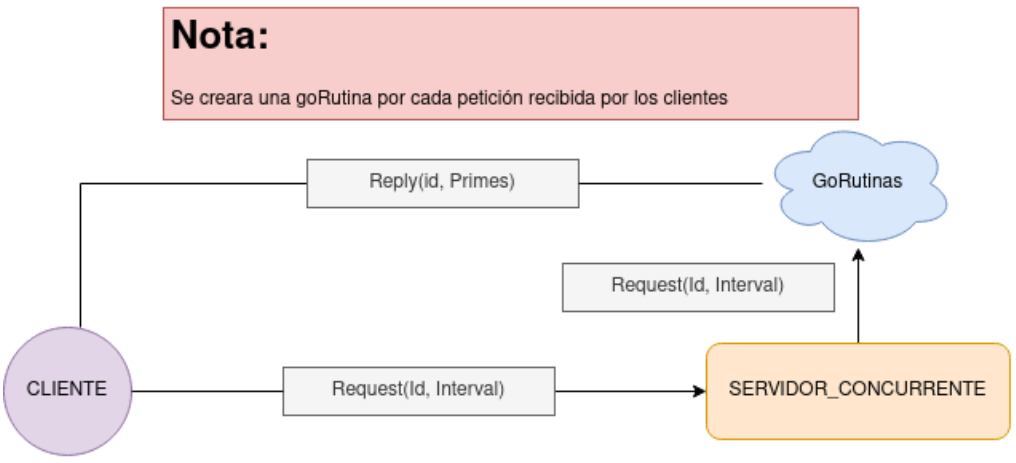
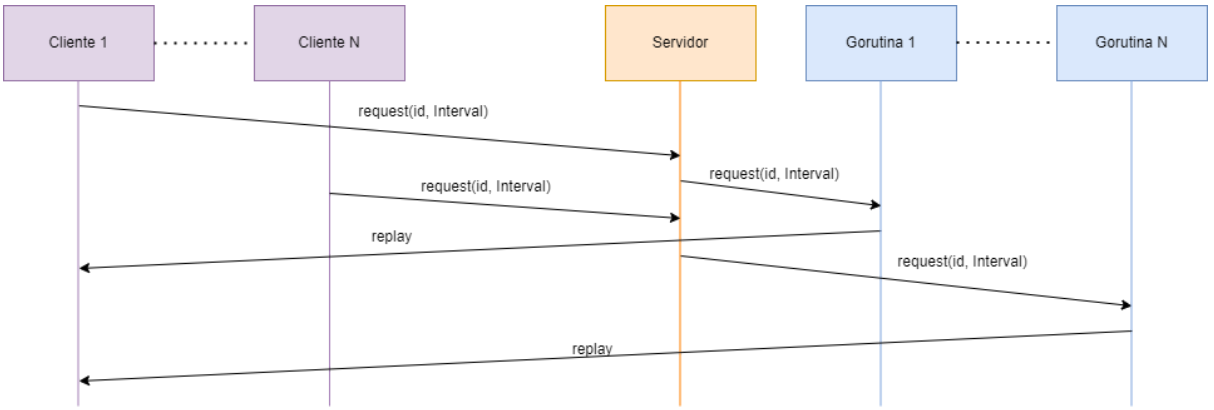


Diagrama de secuencia:



★ **cliente servidor concurrente con un pool fijo de Goroutines**

Mapa de estados:

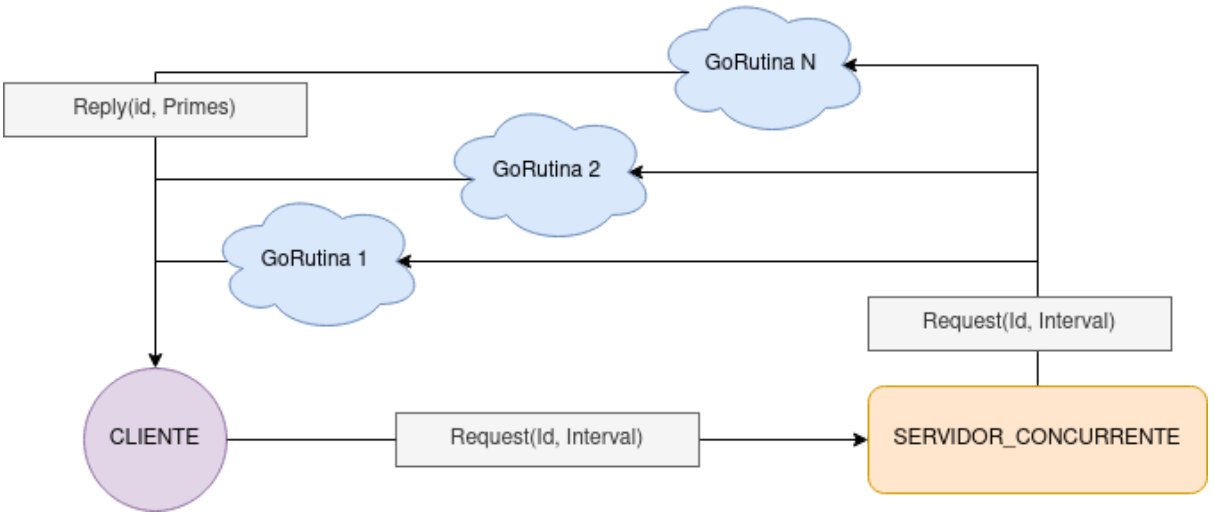
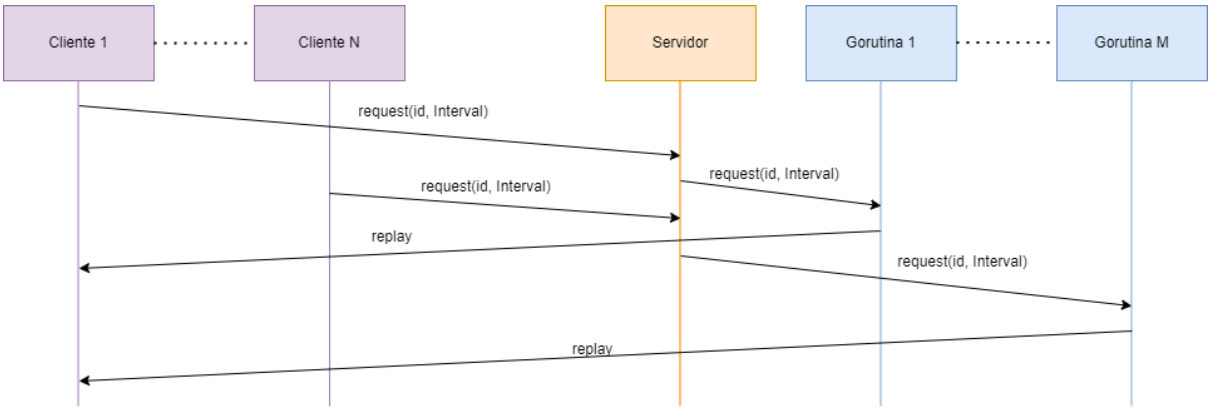


Diagrama de secuencia:



★ **Master-Worker**

Mapa de estados:

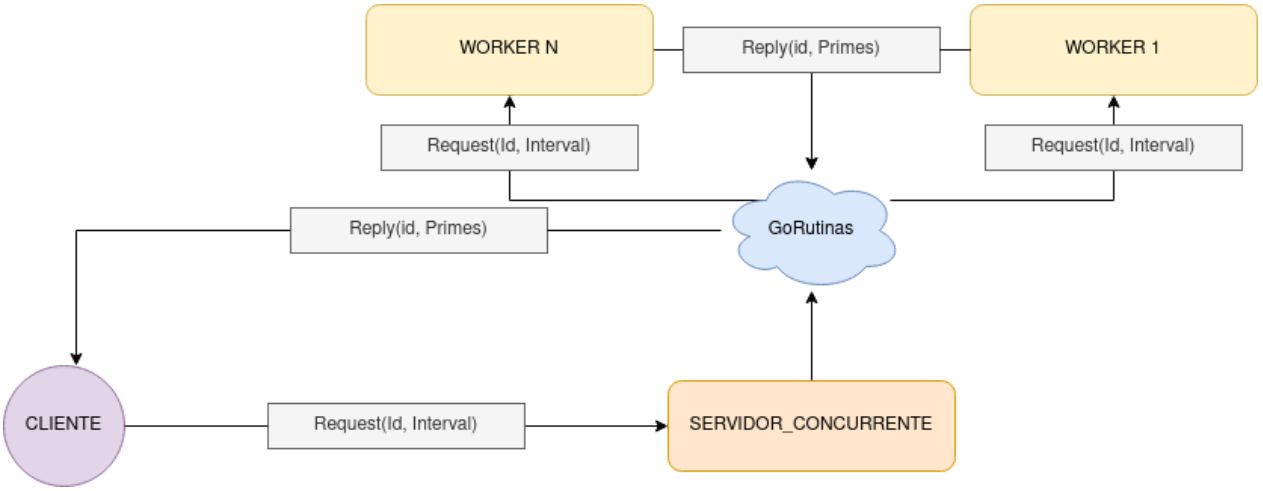
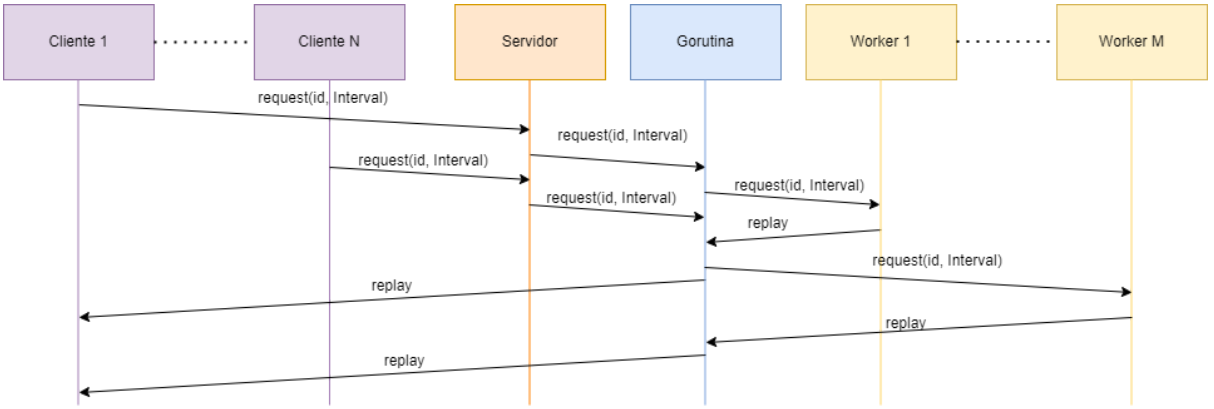


Diagrama de secuencia:



Análisis temporal:

En esta práctica, vamos a considerar el QoS para el cliente, de manera que el tiempo total de ejecución T de una tarea es igual a la suma del t_{ex} (que representa el tiempo de ejecución efectivo en condiciones ideales), más t_{xon} (tiempo de transmisión requerido al intercambiarlos mensajes en la red para realizar la ejecución), más t_o (que es el tiempo de overhead, es decir, los retrasos, las pérdidas y repeticiones de mensajes, todos los imprevistos que puedan surgir).

- Queda como resultado la siguiente fórmula: T = tex + txon + to

- Además buscaremos que se cumpla la siguiente condición:$t_{ex} + t_{xon} + t_o < 2 * t_{ex}$

Para asegurar la veracidad de los resultados se realizará la media de diez ejecuciones para cada uno de los 4 servidores que se han implementado.

Secuencial:

T = (43.5540324 + 44.2217626 + 44.6877605 + 44.4419301 + 44.311260 + 44.7283367 + 44.8691076 + 44.8691076 + 45.5427051 + 45.8724497) / 10 = 44.70984524

Esta arquitectura será la más lenta ya que al ser secuencial no permite realizar varias peticiones del cliente en el mismo instante de tiempo.

Concurrente:

T = (30.1108607 + 30.1236509 + 30.111515 + 30.1161316 + 30.1259836 + 30.101022 + 30.1254804 + 30.106393 + 30.1231289 + 30.1303216)/10 = 30.11744877

En esta arquitectura podemos observar una mejora respecto a la anterior, ya que la concurrencia permite realizar varias peticiones en un mismo instante de tiempo.

Concurrente-pool:

T=(30.137691 + 30.1280425 + 30.0951279 + 30.087523 + 30.0943318 + 30.1315291 + 30.0833821 + 30.0943438 + 30.0819974 + 30.1227287)/10 = 30.10566973

En este caso existe una reducción de tiempo dado que el proceso de crear las goRutinas es alto en tiempo en comparación al tamaño del problema, es por ello que se crea un número fijo de goRutinas. Las cuales permitirán ejecutar varias peticiones del cliente al mismo tiempo sin desperdiciar tiempo creando demasiadas goRutinas de forma innecesaria.

Master-worker:

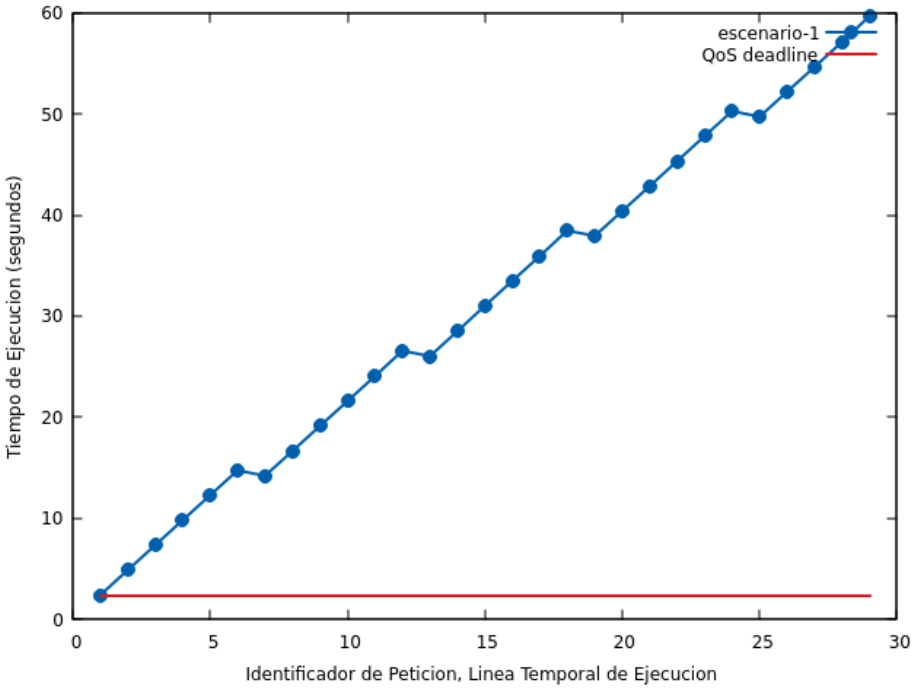
T=(30.1145914 + 30.1201808 + 30.1274548 + 30.1155101 + 30.1172306 + 30.159812 + 30.1076184 + 30.1284735 + 30.1178389 + 30.1179751)/10 = 30.12266856

El tiempo en este caso no implica una gran mejora temporal, pero esto se debe a que se han utilizado un número reducido de workers (2), dado que se ha preferido hacer así para facilitar las pruebas en distribuido y los problemas con los puertos. De todas formas se observa que a pesar de usar tan solo dos trabajadores el tiempo es equitativo al más rápido de los tres que se han ejecutado previamente, obteniendo por ello muy buenos resultados.

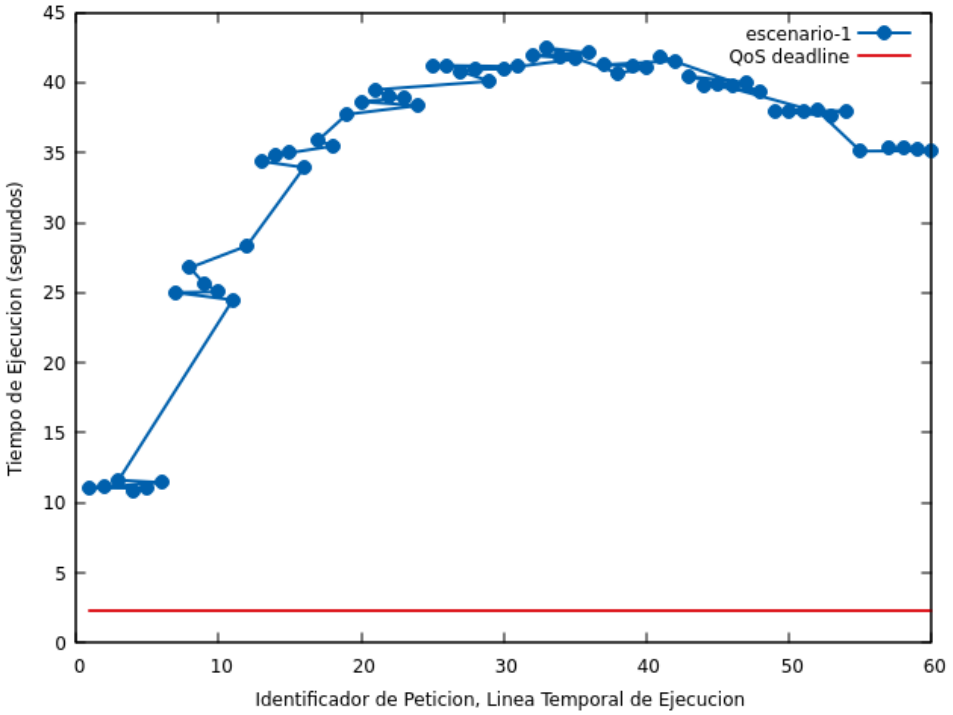
Gnuplot:

Gráficas que permiten visualizar gráficamente las mediciones realizadas previamente.

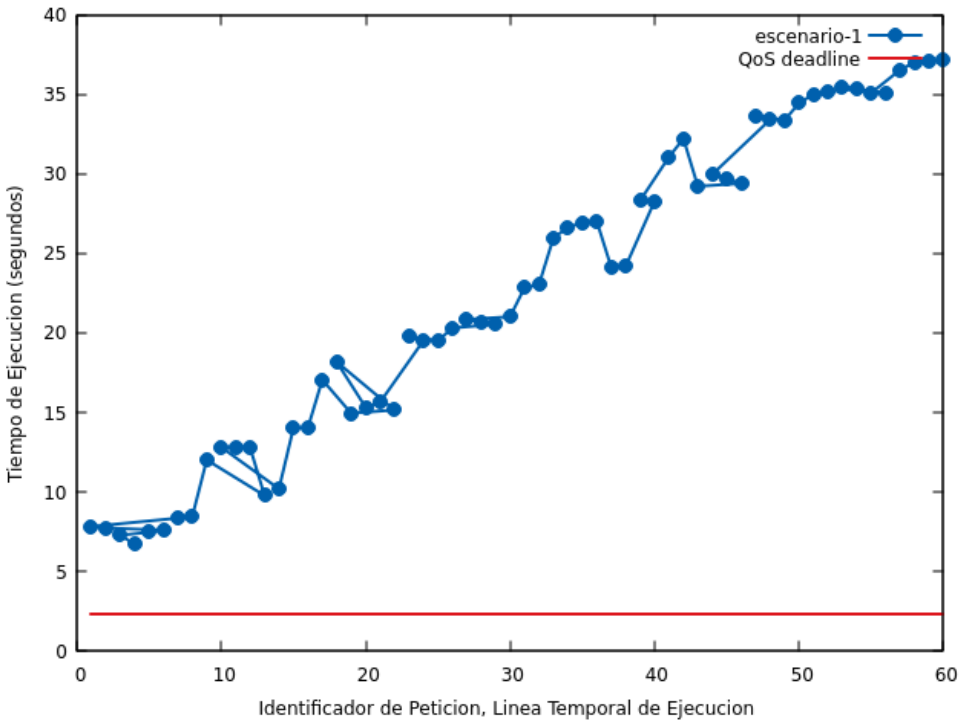
Secuencial



Concurrente1:



Concurriencia pool:



Master-worker