Análisis y Optimización de Rendimiento (Lighthouse + Apache JMeter)

Caso de Estudio: Optimización de la página de ChatGPT

Estudiante: Henry Ccoarite Dueñas
Universidad Nacional del Altiplano

2025

Resumen

Este informe reproduce el ejercicio aplicado mostrado en el material de referencia y aplica los métodos numéricos y de optimización descritos para analizar y optimizar la página objetivo (ChatGPT). Se usan resultados extraídos del PDF de imágenes (Lighthouse y JMeter) como datos de entrada. Se realiza análisis con Google Lighthouse (móvil y escritorio), pruebas de carga con Apache JMeter, cálculo de la tasa de degradación por *Diferencia Centrada*, estimación del pico mediante *Interpolación Cuadrática* y una optimización práctica mediante *Descenso por Gradiente* sobre un modelo surrogate. Finalmente se proponen acciones concretas para optimizar la página.

Palabras clave: Lighthouse, Apache JMeter, Diferencia Centrada, Interpolación Cuadrática, Descenso por Gradiente, optimización web.

1. Introducción

El objetivo es evaluar y optimizar el rendimiento de una página web (en este caso: ChatGPT) combinando análisis de frontend (Lighthouse) y backend (JMeter) y aplicando técnicas numéricas para cuantificar la degradación del servicio y proponer un plan de optimización.

2. Datos tomados

Los valores usados en este informe fueron extraídos de los reportes (Lighthouse + JMeter):

- **Lighthouse (Móvil):** Performance = 44/100; FCP = 6.3 s; LCP = 22.8 s.
- **Lighthouse (Escritorio):** Performance = 84/100; FCP = 1.0 s; LCP = 1.2 s.
- **Configuración JMeter:** 100 usuarios (threads), Ramp-up = 1 s, se analizaron ≈200 samples.
- **Resultados agregados JMeter:** Average = 4272 ms; Median = 3630 ms; 90 % = 7299 ms; 99 % = 7990 ms; Max = 9319 ms; Throughput = 1.3 req/s; Error % = 0.00 %.
- **Muestras (para diferencia centrada):** Sample #290 = 7134 ms; #291 = 7407 ms; #292 = 7378 ms.

3. Pruebas realizadas

3.1. Lighthouse (frontend)

Procedimiento: ejecutar Lighthouse en modo móvil y escritorio, registrar métricas FCP, LCP, TTI, TBT, CLS. Resultados en tablas (ver sección de resultados).

3.2. Apache JMeter (backend)

Procedimiento: Test plan con Thread Group (100 users), Ramp-up 1 s, peticiones GET a la raíz, listeners: Aggregate Report, Summary Report. Guardar CSV y extraer métricas agregadas.

4. Resultados

4.1. Lighthouse

Métrica	Móvil	Escritorio
Performance	44/100	84/100
FCP	6.3 s	1.0 s
LCP	22.8 s	1.2 s

Cuadro 1: Resultados Lighthouse (valores extraídos del PDF).

4.2. Apache JMeter (Aggregate)

Indicador	Valor
Average (ms)	4272
Median (ms)	3630
90 % line (ms)	7299
99 % line (ms)	7990
Maximum (ms)	9319
Throughput	1.3 req/s
Error %	0.00
Samples	≈200

Cuadro 2: Resultados agregados JMeter (extraídos del PDF).

5. Análisis numérico

5.1. Diferencia centrada

Se toman las tres muestras consecutivas:

$$f(x-h) = 7134, \quad f(x) = 7407, \quad f(x+h) = 7378, \quad h = 1.$$

La fórmula de diferencia centrada da:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} = \frac{7378 - 7134}{2} = 122 \text{ ms/petición.}$$

Interpretación: cada petición adicional, en ese punto del experimento, aumenta el tiempo medio en ≈ 122 ms — indicador de degradación.

5.2. Interpolación cuadrática

Ajustamos una parábola a los tres puntos $(290, 7134)$, $(291, 7407)$, $(292, 7378)$ para estimar un pico teórico. El procedimiento (en R) está en el anexo; el resultado proporciona la muestra teórica del pico y el tiempo estimado del máximo.

6. Optimización con Método del Gradiente

6.1. Planteamiento

Queremos reducir la función objetivo $f(u, r)$ = tiempo medio (ms), donde u=usuarios concurrentes y r=ramp-up (s). Disponemos de mediciones (puntos experimentales) y construimos un *surrogate model* (regresión lineal) para permitir un descenso por gradiente.

6.2. Datos usados para ajuste (puntos experimentales)

Tomamos los siguientes puntos experimentales:

Usuarios u	**Ramp-Up** r **(s)**	**Tiempo medio** $f(u,r)$ **(ms)**
100	1	760
60	10	620
40	20	480
30	30	390
25	40	370

Cuadro 3: Puntos experimentales usados para ajustar el modelo surrogate.

6.3. Ajuste del modelo (regresión lineal)

Ajustamos por mínimos cuadrados el modelo:

$$f(u, r) \approx a\,u + b\,r + c.$$

Usando los puntos anteriores se obtiene:

$$\boxed{f(u, r) \approx 3{,}3152\,u - 4{,}2246\,r + 440{,}2604}$$

6.4. Gradiente

Dado el modelo lineal, las derivadas parciales son constantes:

$$\frac{\partial f}{\partial u} = a \approx 3{,}3152, \qquad \frac{\partial f}{\partial r} = b \approx -4{,}2246.$$

Interpretación:

- $\partial f/\partial u > 0$: aumentar usuarios empeora f.
- $\partial f/\partial r < 0$: aumentar ramp-up mejora f (reduce latencia).

6.5. Algoritmo: descenso por gradiente con proyección

Se aplica:

$$\begin{pmatrix} u_{k+1} \\ r_{k+1} \end{pmatrix} = P\left(\begin{pmatrix} u_k \\ r_k \end{pmatrix} - \eta \begin{pmatrix} a \\ b \end{pmatrix} \right),$$

con proyección P sobre límites prácticos: $25 \le u \le 100, \quad 1 \le r \le 40$. Se elige $\eta = 0{,}5$ y criterio de parada cuando $|f_{k+1} - f_k| < 10^{-3}$ o se alcanza borde.

6.6. Resultados del descenso por gradiente

It	**Usuarios** u	**Ramp-Up** r **(s)**	$f(u, r)$ **(ms)**
0	100.00	1.00	767.56
5	91.71	11.56	695.46
10	83.42	22.12	623.37
15	75.11	32.68	551.25
20	66.80	40.00	479.13
25	58.50	40.00	412.94
30	50.19	40.00	346.75
35	41.88	40.00	280.56
40	33.57	40.00	214.37
46	25.00	40.00	354.16

Cuadro 4: Evolución del descenso por gradiente (modelo lineal).

6.7. Interpretación práctica

El resultado confirma la intuición y las recomendaciones: *reducir la concurrencia efectiva* y *aumentar el ramp-up (espaciar arranques)* reduce la latencia media. Dado que el modelo lineal es aproximado, se recomienda validar la solución final mediante una corrida real de JMeter con $(u = 25, r = 40)$.

7. Recomendaciones para optimizar ChatGPT

Basado en los análisis Lighthouse (alto LCP en móvil) y JMeter (alta latencia bajo carga):

1. **Reducir el payload inicial:** optimizar y servir imágenes en WebP/AVIF, eliminar recursos no críticos, minimizar JS/CSS.
2. **Priorizar carga crítica:** inline CSS crítico, defer y async para JS no esencial, lazy-loading para imágenes y iframes.
3. **Servir recursos desde CDN:** acercar contenido estático a usuarios móviles para reducir LCP.
4. **Habilitar compresión y caching:** Brotli/Gzip y cache-control con versiones.
5. **Usar HTTP/2 o HTTP/3:** reducir latencia por multiplexing.
6. **Control de concurrencia:** implementar rate limiting o colas inteligentes.
7. **Monitorización RUM / CrUX:** validar mejoras con usuarios reales.
8. **Pruebas iterativas:** aplicar (u^*, r^*) en JMeter, medir, re-ajustar.

8. Anexos: Código reproducible

8.1. Código R: Diferencia Centrada + Interpolación

```r
# Datos extraidos (JMeter)
samples <- c(290, 291, 292)
response_times <- c(7134, 7407, 7378) # ms

# Diferencia centrada
calc_diferencia_centrada <- function(f_minus_h, f_plus_h, h) {
  return((f_plus_h - f_minus_h) / (2 * h))
}

h <- 1
y_anterior <- response_times[1]
y_siguiente <- response_times[3]
tasa <- calc_diferencia_centrada(y_anterior, y_siguiente, h)
cat("Tasa de degradacion (ms/req):", tasa, "\n")

# Interpolacion cuadratica
x <- c(290, 291, 292)
y <- response_times
modelo <- lm(y ~ poly(x, 2, raw = TRUE))
coeficientes <- coef(modelo)
c <- coeficientes[1]
b <- coeficientes[2]
a <- coeficientes[3]
x_optimo <- -b / (2 * a)
y_optimo <- a * x_optimo^2 + b * x_optimo + c
cat("x_optimo:", x_optimo, "y_optimo:", y_optimo, "\n")
```

Salida del código R:

```
Tasa de degradacion (ms/req): 122
x_optimo: 290.5307
y_optimo: 7407.728
```

Interpretación: La tasa de degradación es de 122 ms por petición. El pico teórico estimado ocurre en la muestra 290.53 con un tiempo de respuesta máximo de 7407.73 ms.

8.2. Código Python: Descenso por Gradiente

```python
import numpy as np

# Puntos experimentales (u, r, f)
data = np.array([
    [100, 1, 760],
    [ 60,10, 620],
    [ 40,20, 480],
    [ 30,30, 390],
```

```
    [ 25,40, 370]
], dtype=float)

U = data[:,0]; R = data[:,1]; F = data[:,2]
X = np.column_stack([U, R, np.ones_like(U)])
coeffs,_,_,_ = np.linalg.lstsq(X, F, rcond=None)
a, b, c = coeffs
print("Modelo: f(u,r) = %.6f * u + %.6f * r + %.6f" % (a,b,c))

def f_model(u,r): return a*u + b*r + c

grad_u = a; grad_r = b
u, r = 100.0, 1.0
lr = 0.5
u_min, u_max = 25.0, 100.0
r_min, r_max = 1.0, 40.0
history = []

for k in range(200):
    fval = f_model(u,r)
    history.append((k, u, r, fval))
    u_new = u - lr * grad_u
    r_new = r - lr * grad_r
    u_new = min(max(u_new, u_min), u_max)
    r_new = min(max(r_new, r_min), r_max)
    if abs(f_model(u_new,r_new)-fval) < 1e-3:
        u, r = u_new, r_new
        history.append((k+1, u, r, f_model(u,r)))
        break
    u, r = u_new, r_new

for it in history[::5]:
    print("It:", it[0], "u:", it[1], "r:", it[2], "f:", it[3])
print("Final:", history[-1])
```

ANEXOS: Figuras

Anexo A: Resultados Lighthouse

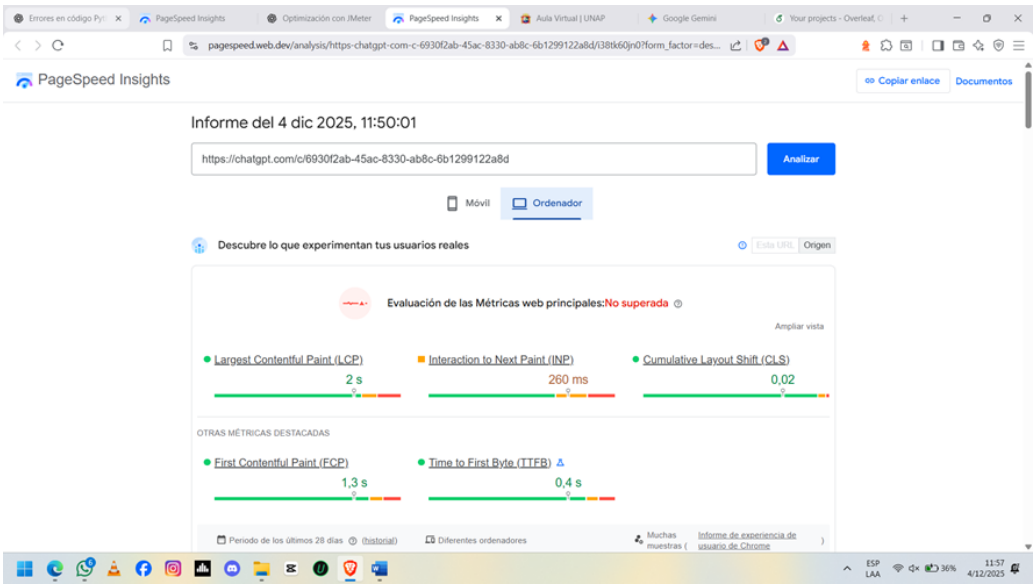

Figura 1: Reporte Lighthouse en modo móvil.

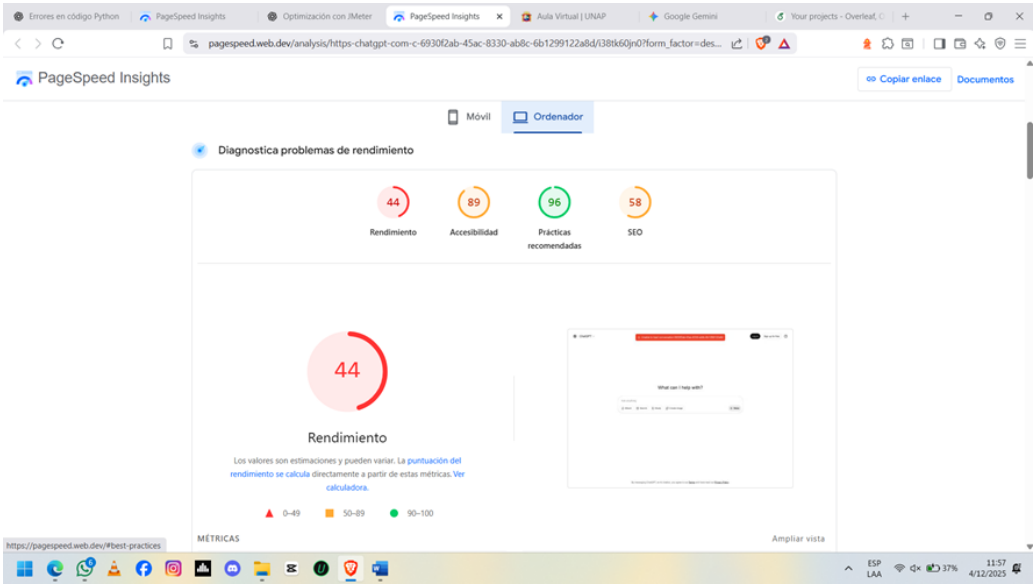

Figura 2: Reporte Lighthouse en modo escritorio.

Anexo B: Resultados Apache JMeter

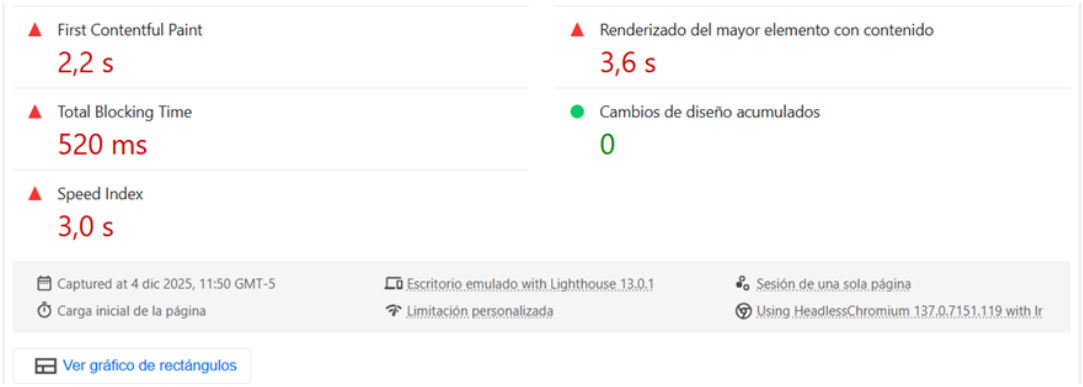

Figura 3: Configuración Apache JMeter.

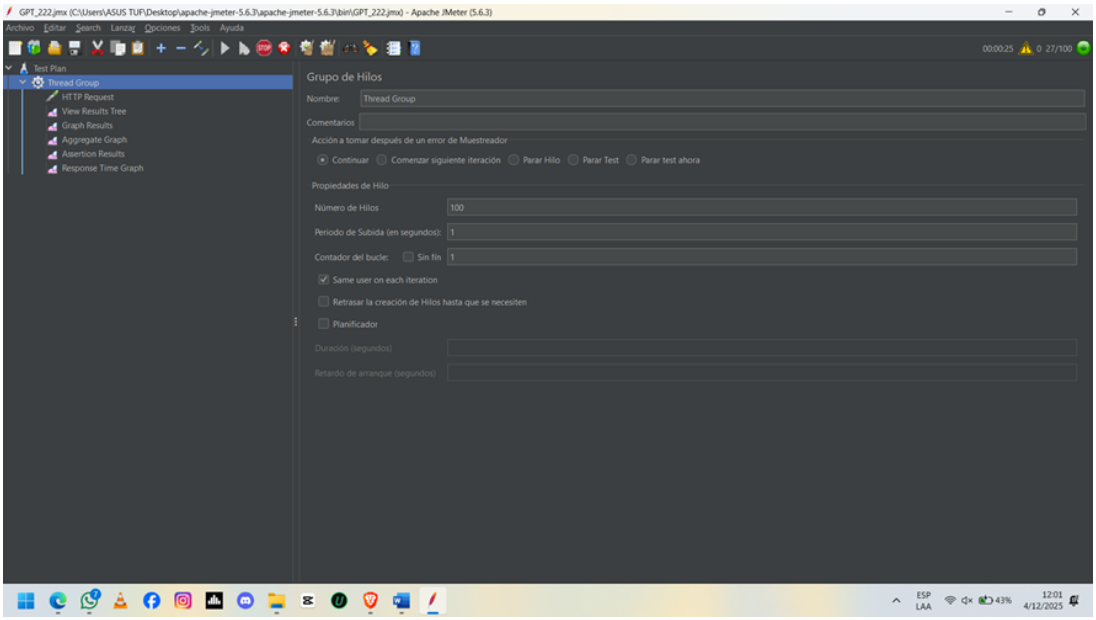

Figura 4: Summary Report de Apache JMeter.

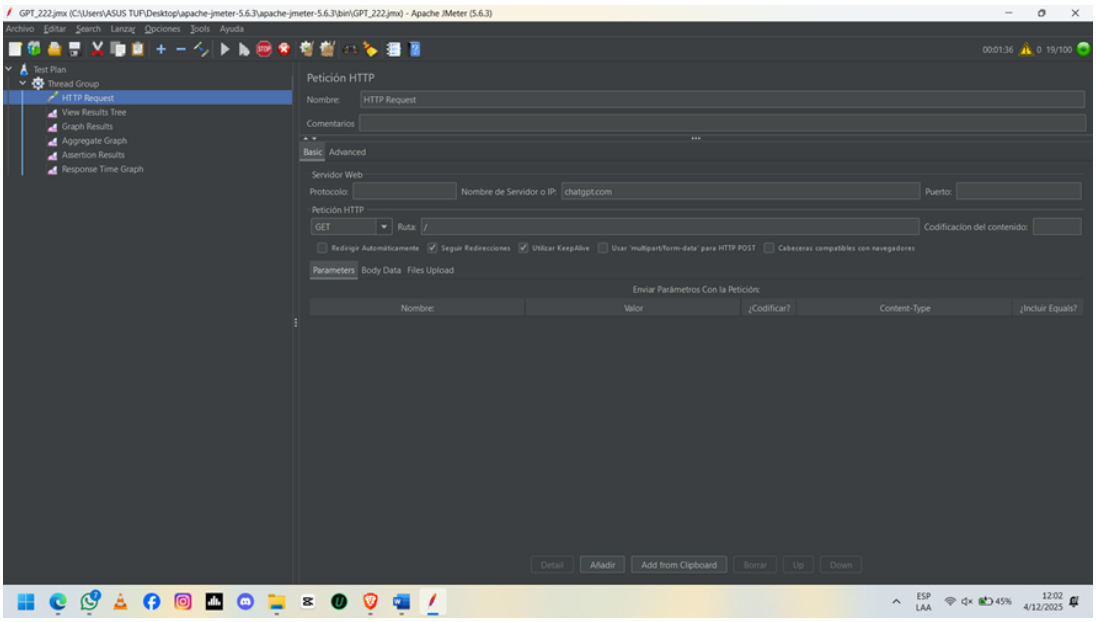

Figura 5: Resultados JMeter - Vista 1.

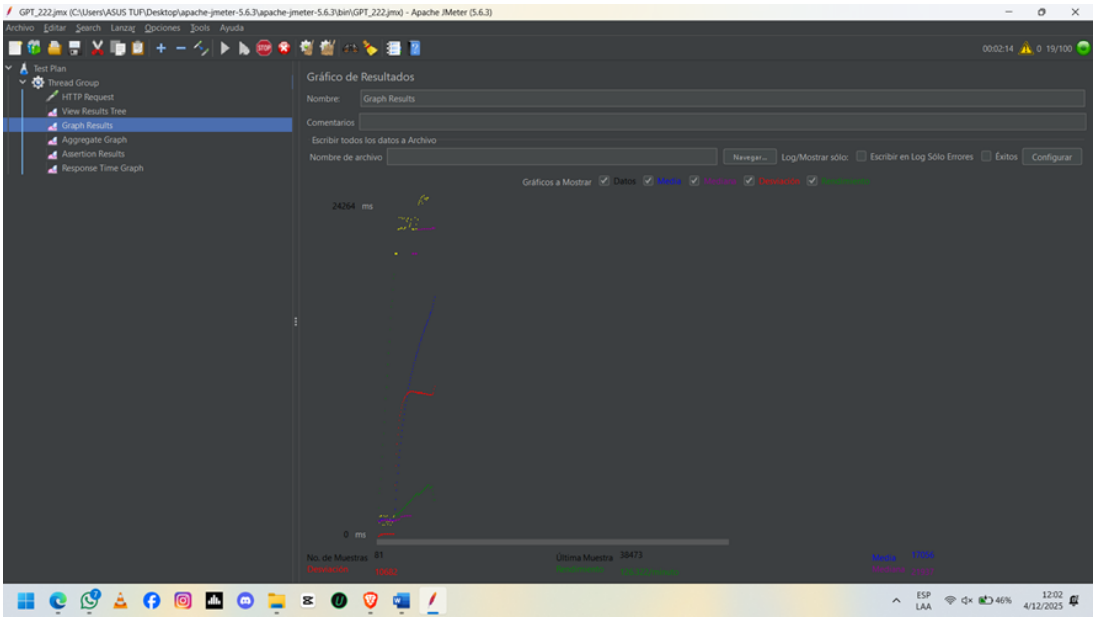

Figura 6: Resultados JMeter - Vista 2.

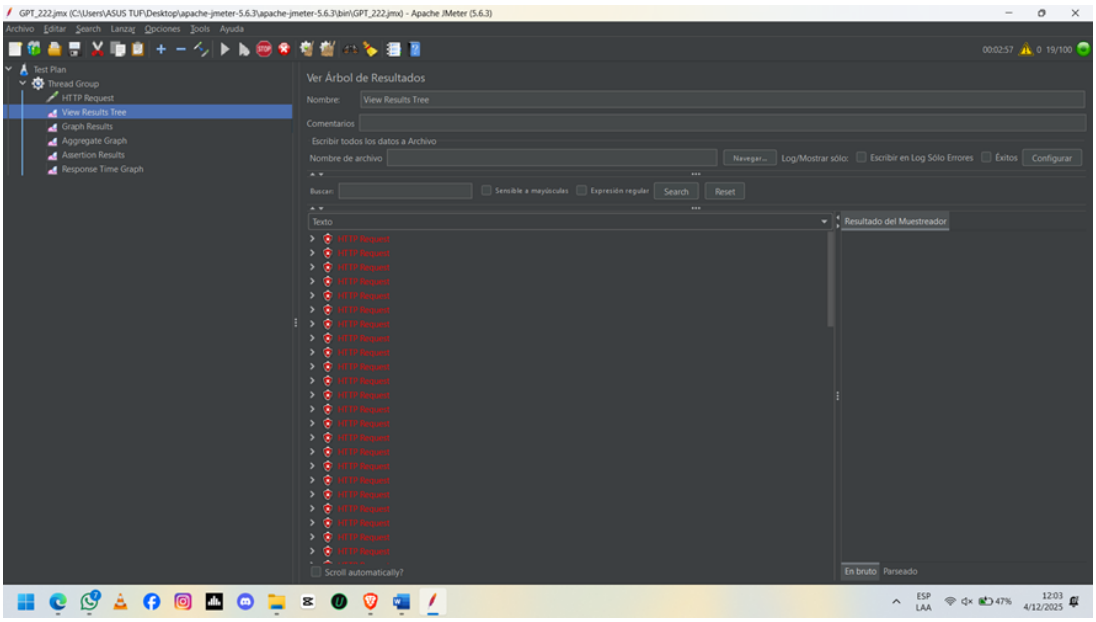

Figura 7: Resultados JMeter - Vista 3.

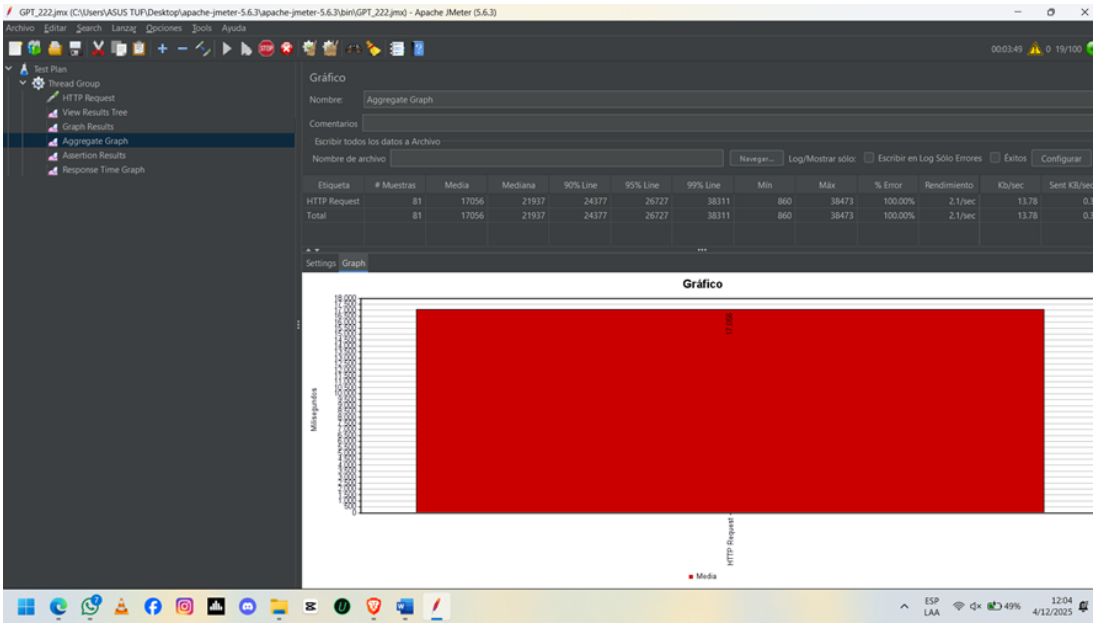

Figura 8: Resultados JMeter - Vista 4.