Mauricio Neira 201424001
Daniel Agudelo 201533411

Caso de estudio 3

Infraestructura computacional

1. *Descripción detallada de los cambios realizados para medir los indicadores, incluyendo el código (cliente o servidor) y líneas dónde se hizo el cambio*

Primero, lo que se hizo, a diferencia del caso anterior, fue separar el Main del Cliente, dado que ahora el Main actuaría como generador para GLoad.

- Para medir el tiempo de creación de la llave simétrica, se toma el tiempo desde el comienzo de la etapa 4. Se declara y asigna una variable tiempoActual así:

```
long tiempoActual = System.currentTimeMillis();
```

Luego, después de la sentencia que crea la llave simétrica, es decir:

```
simmetricKey = new SecretKeySpec( clearText, 0, clearText.length, sim );
```

Se asigna la varible de tiempoCreacionLlaves así:

```
tiempoCreacionLlaves = System.currentTimeMillis() - tiempoActual;
```

- Para medir el tiempo de actualización, se toma el tiempo desde que el escritor manda ACT1, es decir:

Se asigna la variable sToSend, que es lo que se enviará:

```
sToSend = "ACT1:"+sToSend;
```

Luego, se toma el tiempo:

```
tiempoActual = System.currentTimeMillis();
```

Y se envía la actualización

```
escritor.println(sToSend);
```

, luego se hace el hash del mensaje para enviar, ACT2, se envía ACT2, usando mla misma variable sToSend, y se toma el tiempo:

```
escritor.println(sToSend);
timepoACT1 = System.currentTimeMillis() - tiempoActual;
```

- Para medir el número de transacciones perdidas, antes de correr el programa se tiene la información de cuántas transacciones son. Llame t a ese número. Para recolectar la información de los anteriores indicadores, cada vez que termine una transacción exitosamente, se imprimen los datos en un archivo (tiempoCreacionLlaves, tiempoACT1, cpuUsage), pero no de lo contrario. Así, para obtener el número de transacciones perdidas, basta con restar de t, el número de registros en el archivo.

- Para medir el uso de la CPU, se usa el método getProcessCPULoad(), en el servidor, al terminar la etapa 4 (es la última sentencia), y se le envía al cliente:

```
write(writer,getProcessCpuLoad() + "");
```

En el cliente, también como última sentencia, se asigna a la variable cpuUsage lo que llegue del Servidor:

```
cpuUsage = Double.parseDouble(lector.readLine());
```

2. *Identificación de la plataforma:*

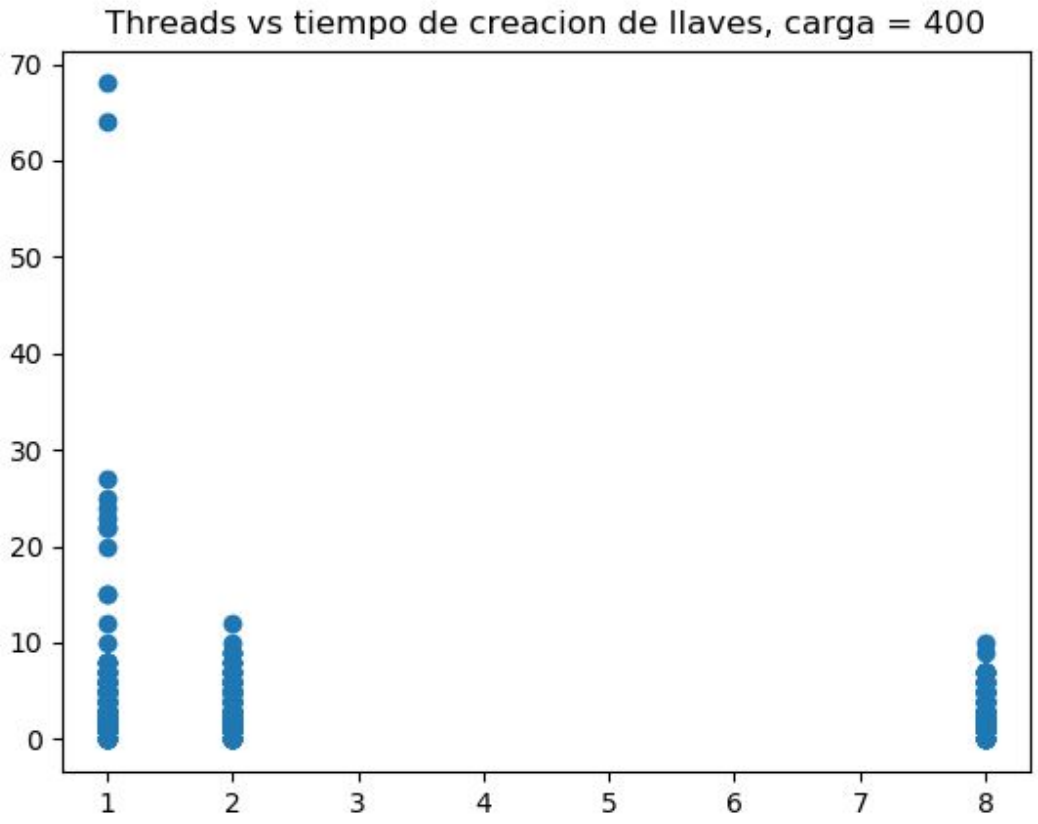
Ambas máqunas son de 64 bits con 8Gb de RAM . 4 núcleos físicos (8 virtuales)

3. *Respuestas al punto 3 (todas las gráficas y los comentarios)*

a. Fije Carga y genere: # threads vs. tiempo de creación de la llave (3 cargas diferentes = 3 gráficas, 3 tamaños de pool -1, 2 y 8-: 3 conjuntos de puntos por gráfica):

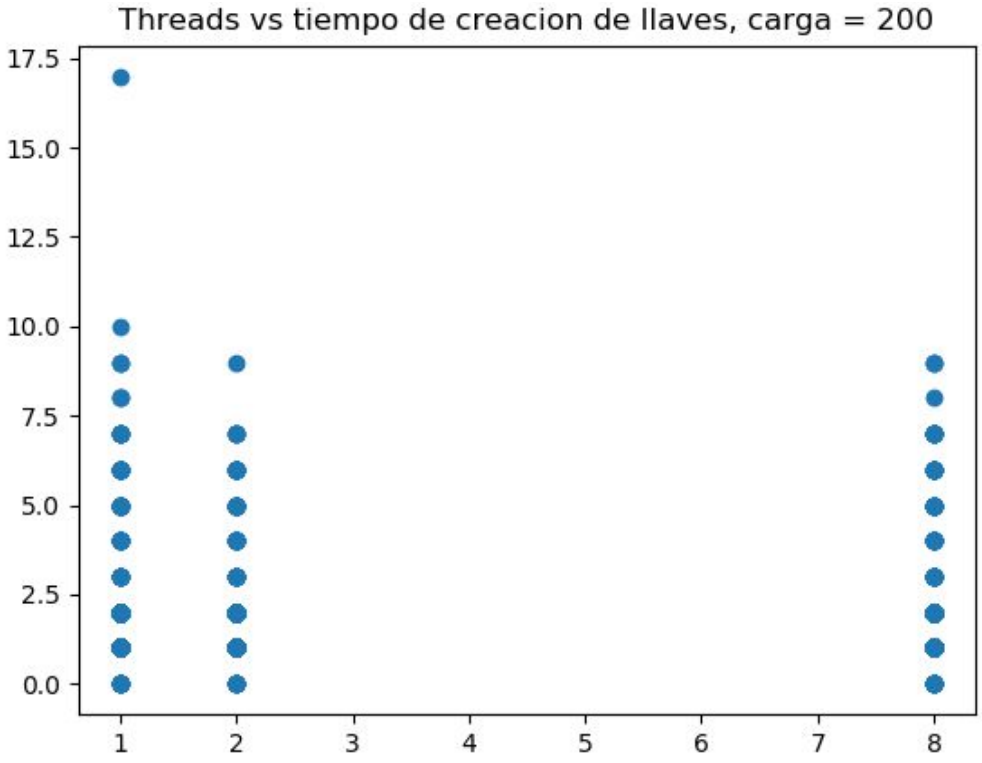
Carga 400, 20ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411

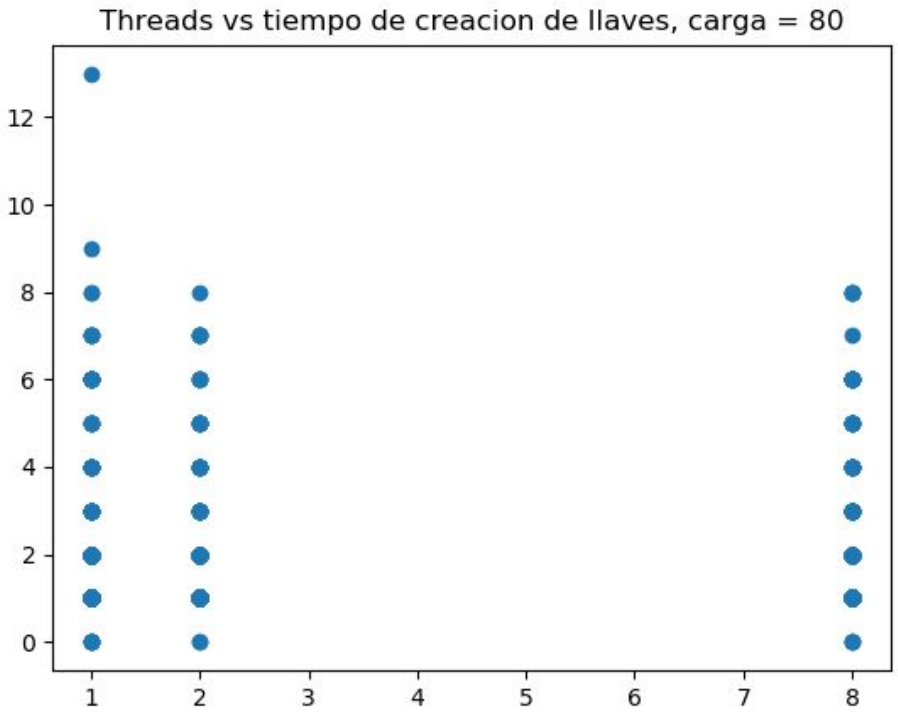


Carga 200, 40 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411



Carga 80, 100 ms de retardo:

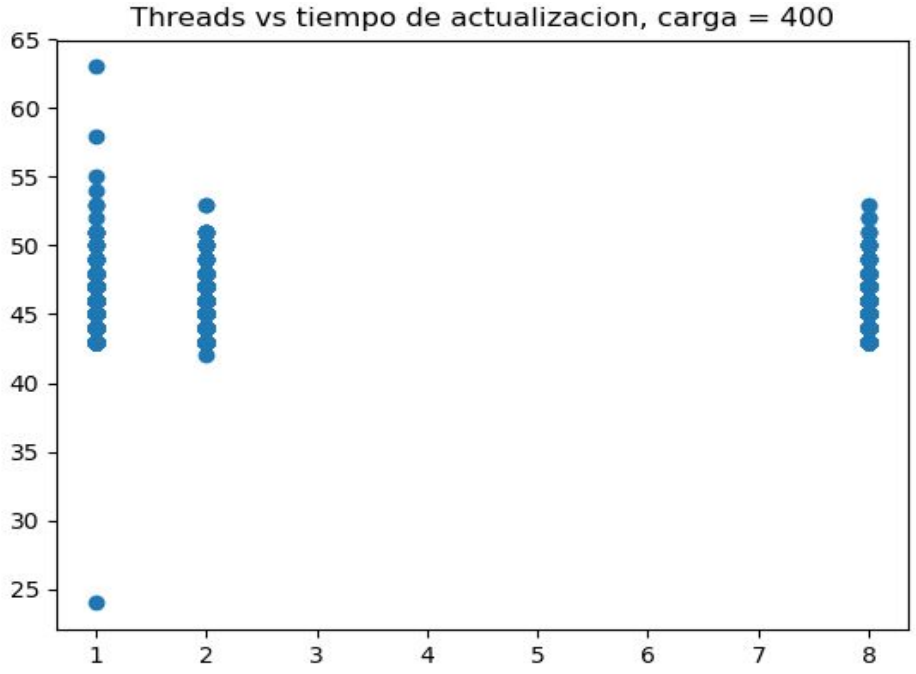


Aquí el comportamiento es el esperado para el aumento de carga. Bajo más carga se tiene, hay un aumento en el tiempo de la generación de llaves. Sin embargo, hay resultados curiosos, 1 solo thread se demora mas generando la llave que 2 u 8 threads. Esto es contraintuitivo pues con mas threads, hay más peticiones siendo generadas en paralelo. Debería ser al revés.

Sin embargo, adentrándonos en la implementación de Gload (que es una implementación un poco pobre, debo resaltar), TODOS los threads se lanzan al mismo tiempo. Es decir 8+2+1 = 11 threads están activos en todo momento. Con muchos threads, el total de la carga se consume rápidamente mientras que 1 thread tiene que generar la totalidad de la carga. Así pues, ese thread dura activo el tiempo que duran activos los otros 10 y esto genera demoras pues se están compartiendo muchos recursos en la máquina.

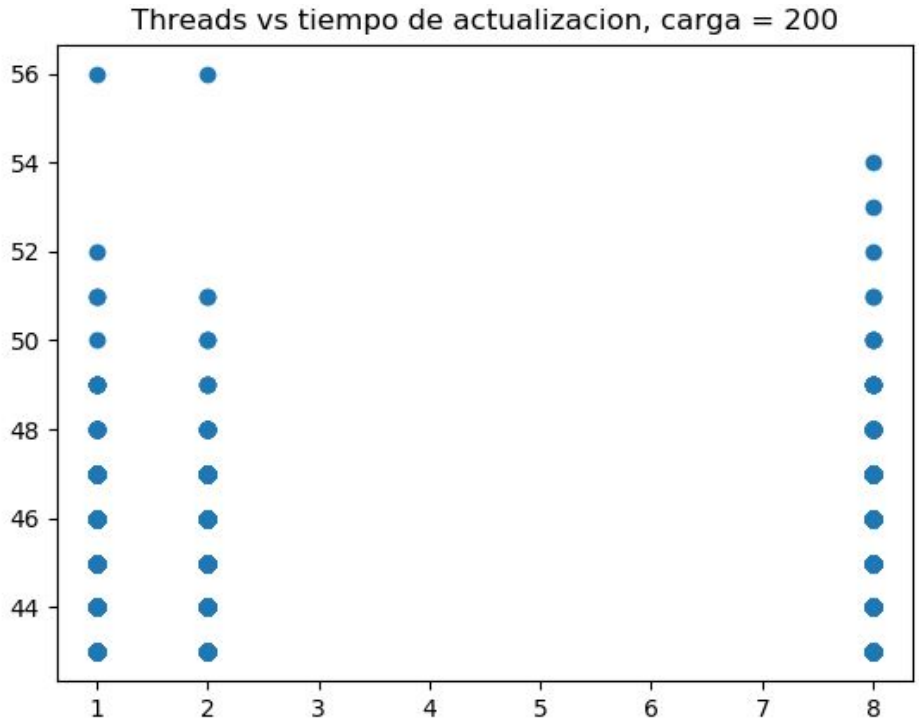
b. Fije Carga y genere: # threads vs. tiempo de actualización (mismo número de gráficas):

Carga 400, 20 ms de retardo:

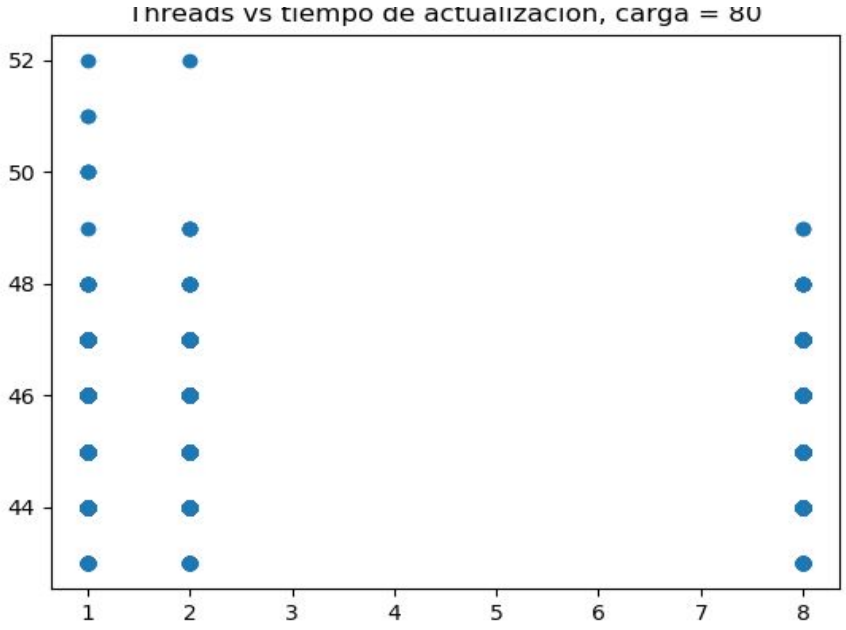


Carga 200, 40 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411



Carga 80, 100 ms de retardo:

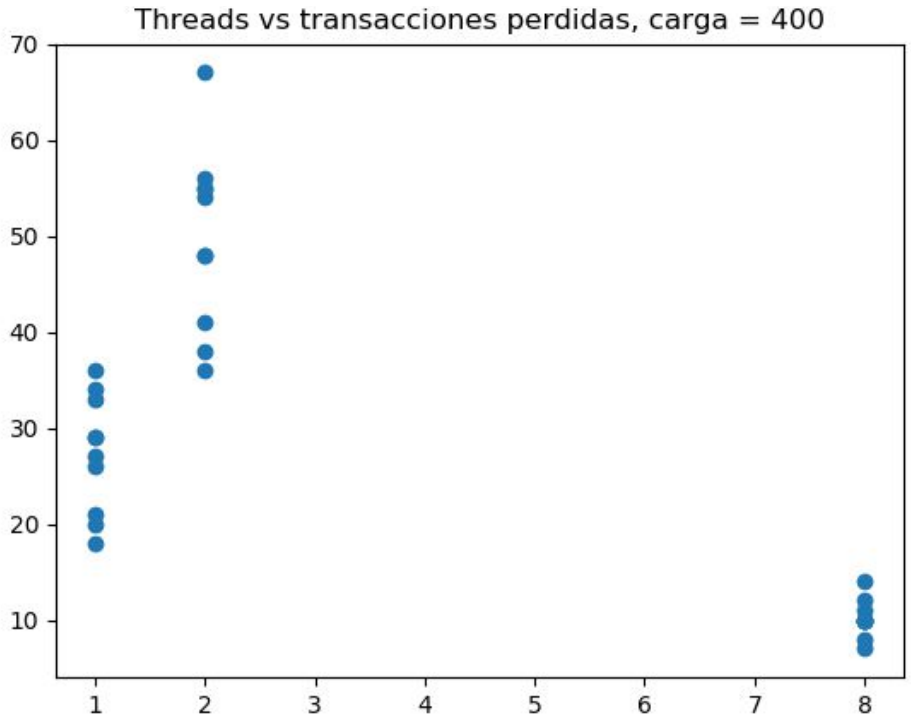


Se observa el mismo comportamiento que en el punto a.

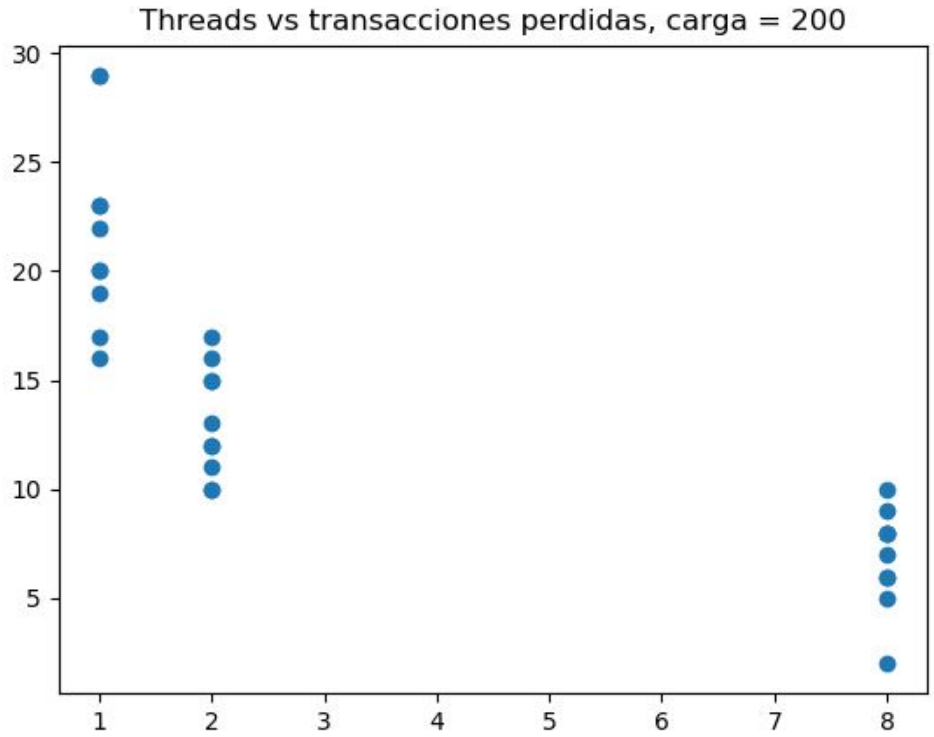
c. Fije Carga y genere: # threads vs. # de transacciones perdidas (mismo número de gráficas):

Mauricio Neira 201424001
Daniel Agudelo 201533411

Carga 400, 20 ms de retardo:

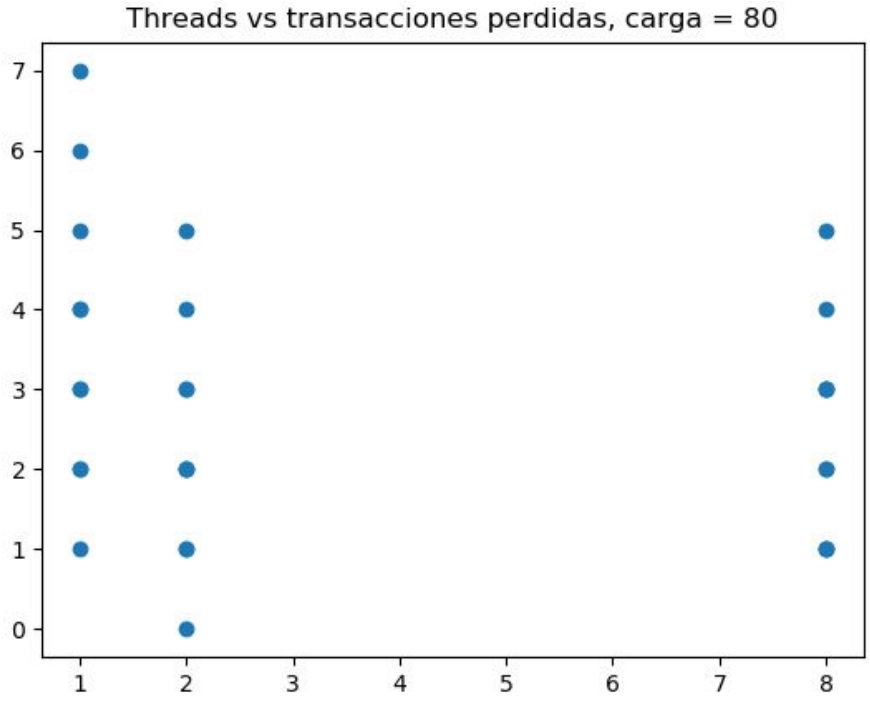


Carga 200, 40 ms de retardo:



Mauricio Neira 201424001
Daniel Agudelo 201533411

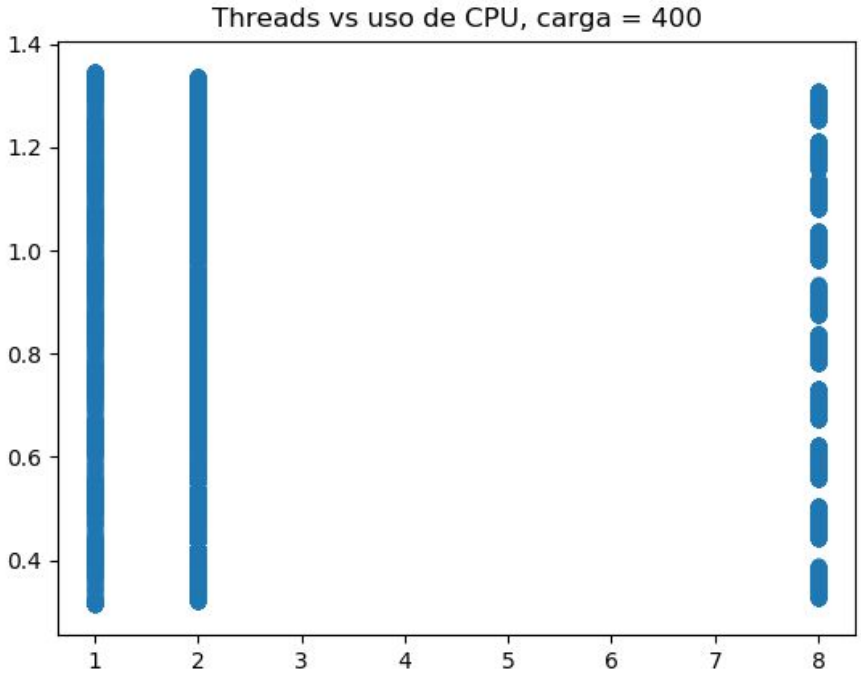
Carga 80, 100 ms de retardo:



Se observa el mismo comportamiento que en el punto a pero adicionalmente, hay factores estocásticos que se tienen que tomar en cuenta. Es por esto que en algunos casos 2 threads son mejores que 1 y vice-versa.

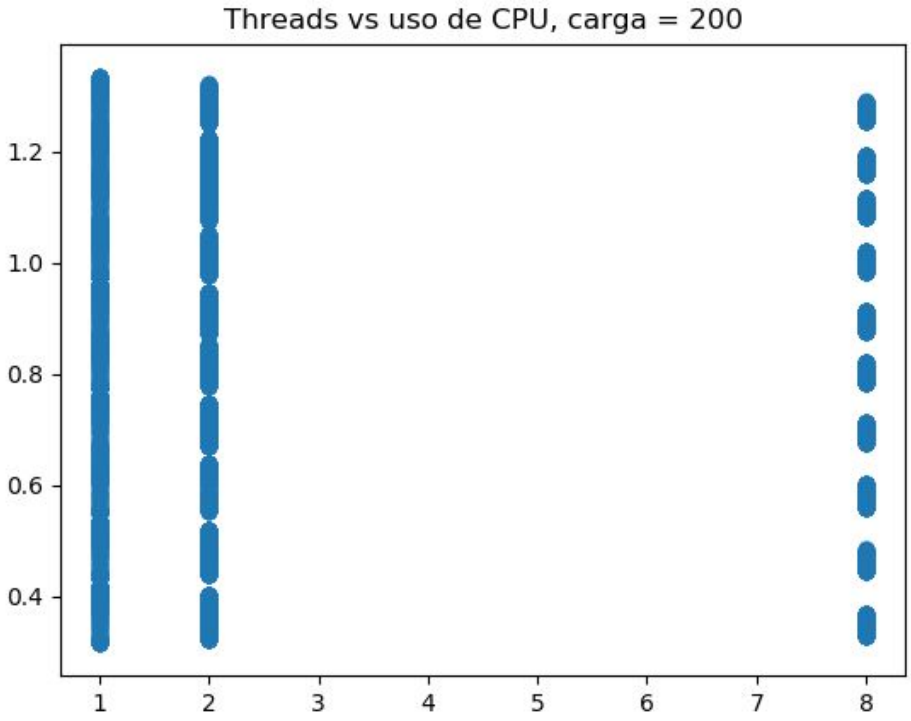
d. Fije Carga y genere # threads vs. porcentaje de uso de la CPU en el servidor (mismo número de gráficas):
Carga 400, 20 ms de retardo

Mauricio Neira 201424001
Daniel Agudelo 201533411

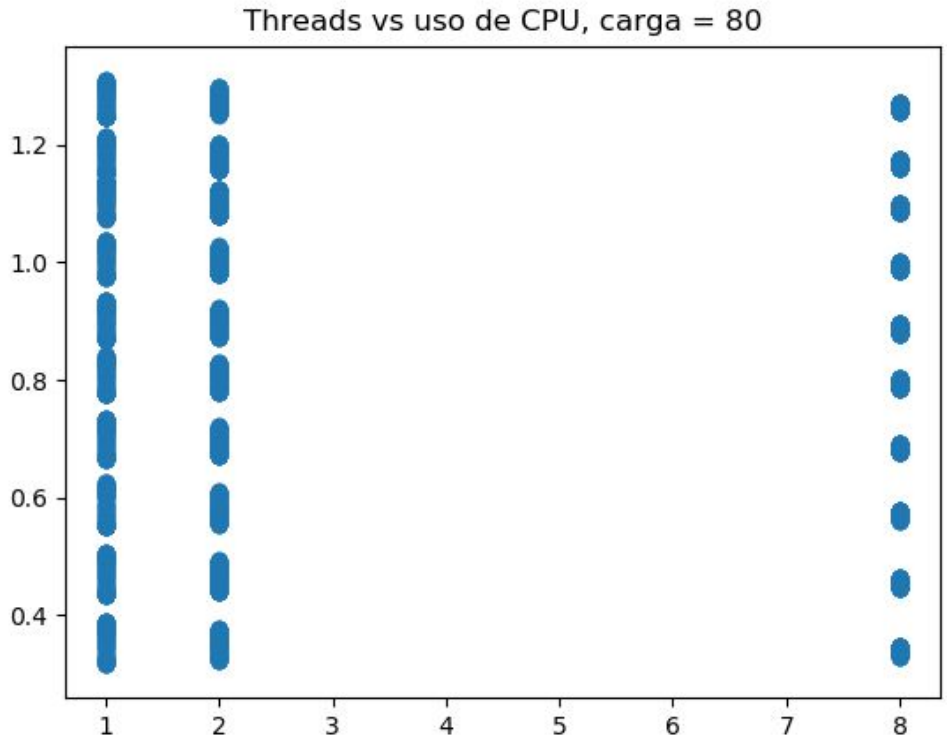


Mauricio Neira 201424001
Daniel Agudelo 201533411

Carga 200, 40 ms de retardo:



Carga 80, 100 ms de retardo:



Aquí se muestra como en todos los casos, todos los threads estan repartidos uniformemente en términos de uso de CPU. Esto respalda el hecho queGLoad lanza TODOS los threads al mismo tiempo. Todos empiezan utilizando pocos recursos de la CPU, a medida que varios se ejecutan, van aumentando el uso de la CPU hasta que llega a un máximo local. Una vez terminan los threads, el uso de la CPU baja. Este barrido ocupa valores desde 0 hasta el máximo local. Así pues, se tienen los resultados observados.

Cabe notar que para carga 400, el máximo uso de CPU aumenta ligeramente. Esto tiene sentido pues en total, la CPU está bajo más estrés.

4. ***Respuestas al punto 4 (todas las gráficas y los comentarios)***

4.1. Repita los experimentos del punto 3 (para los indicadores 2, 3 y porcentaje de uso de CPU), usando un servidor y un cliente sin seguridad.

4.2. Responda: ¿Cuál es el resultado esperado sobre el comportamiento de una aplicación que implemente funciones de seguridad vs. una aplicación que no implementa funciones de seguridad?

En el caso con seguridad, el Servidor tiene que generar la llave simétrica, cifrarla para poder enviarla protegida y hacer esto último. El Cliente, por su parte, debe recibir el mensaje, descifrarlo para poder obtener la llave simétrica, cifrar sus coordenadas con esa llave, y enviar su actualización.
En el caso sin seguridad, el Servidor sólo pide la actualización (con el mensaje de INICIO) y el Cliente sólo tiene que enviar los bytes de sus coordenadas sin cifrar.

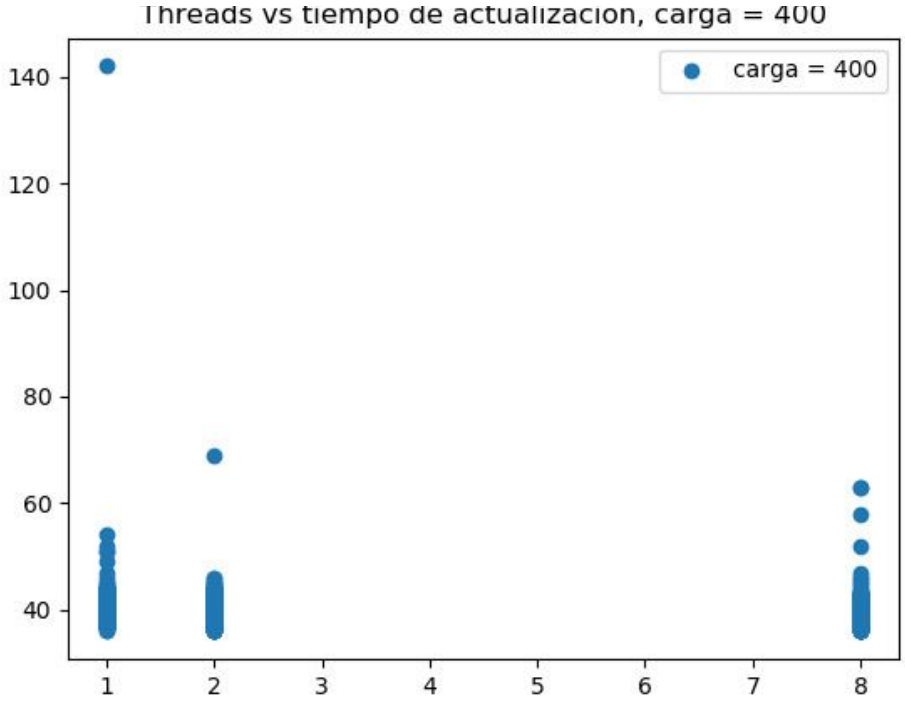
Podemos apreciar entonces, que el tiempo de creación de la llave simétrica y del cifrado del mensaje se ahorran en el caso sin seguridad, por lo que podemos esperar que este último sea más rápido, en la medida en que los tiempos de creación de llaves son 0. Además, podemos esperar menos errores (menos transacciones perdidas): En el caso con seguridad, ocurrió que los mensajes cifrados que contenían la llave simétrica no llegaron bien, y por lo tanto el cifrado que usaba esa llave se veía afectado.

4.3. Genere las gráficas b, c y d del punto 3, e indique si ellas confirman los resultados esperados (en el punto anterior). Justifique su respuesta.

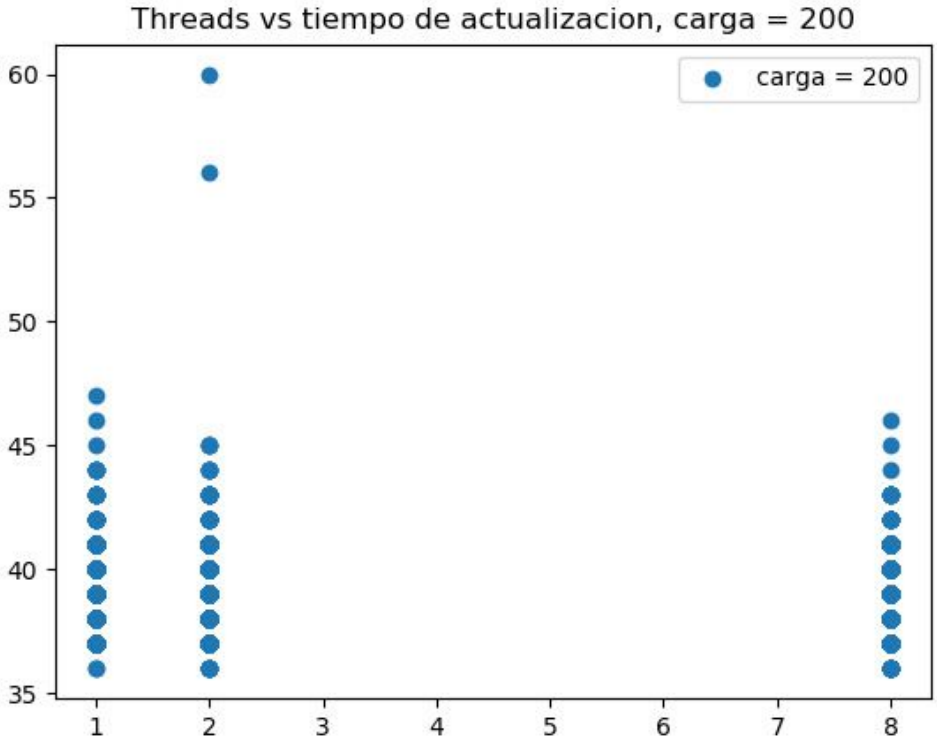
- **Tiempo de Actualización:**

Mauricio Neira 201424001
Daniel Agudelo 201533411

Carga 400, 20 ms de retardo:

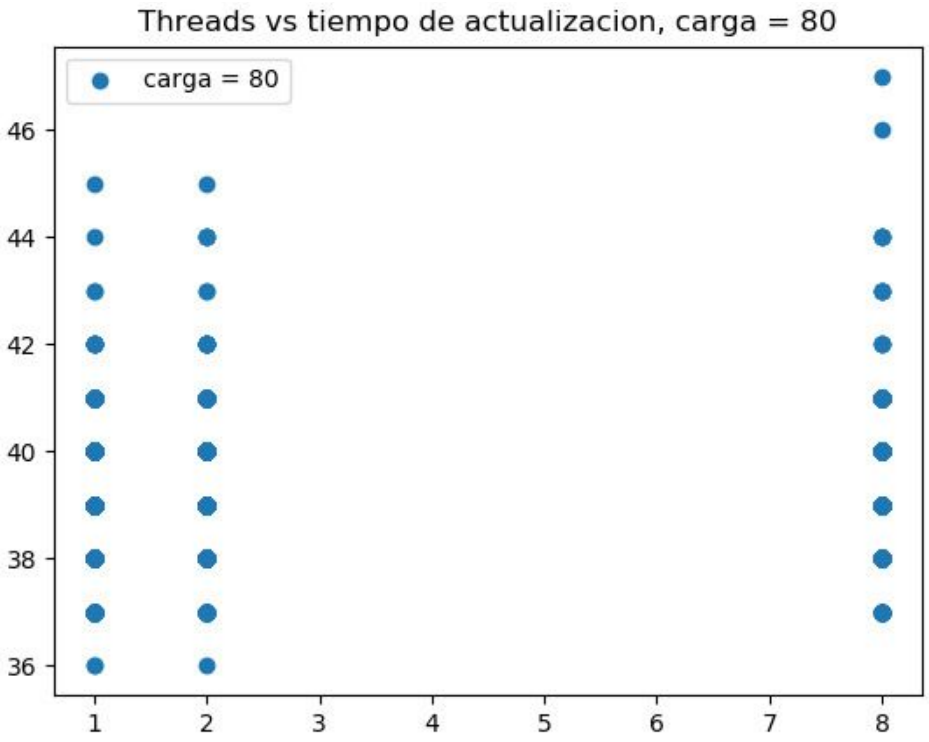


Carga 200, 40 ms de retardo:



Carga 80, 100 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411



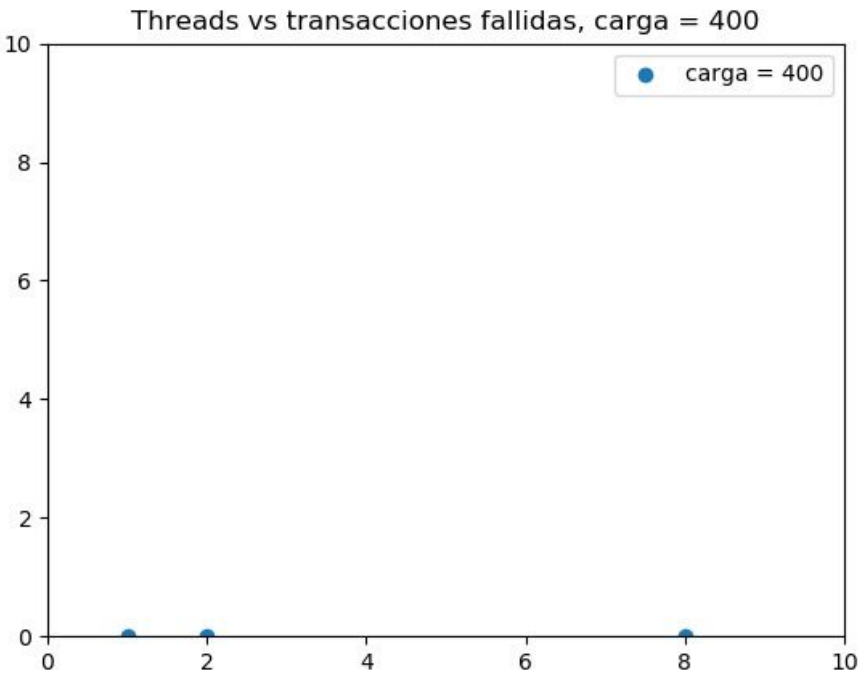
En el caso del servidor sin seguridad, se observa una reducción en el tiempo de actualización pues ya no se necesita tanto tiempo de cómputo para correr cada unidad de carga.

Esto confirma las hipótesis estipuladas.

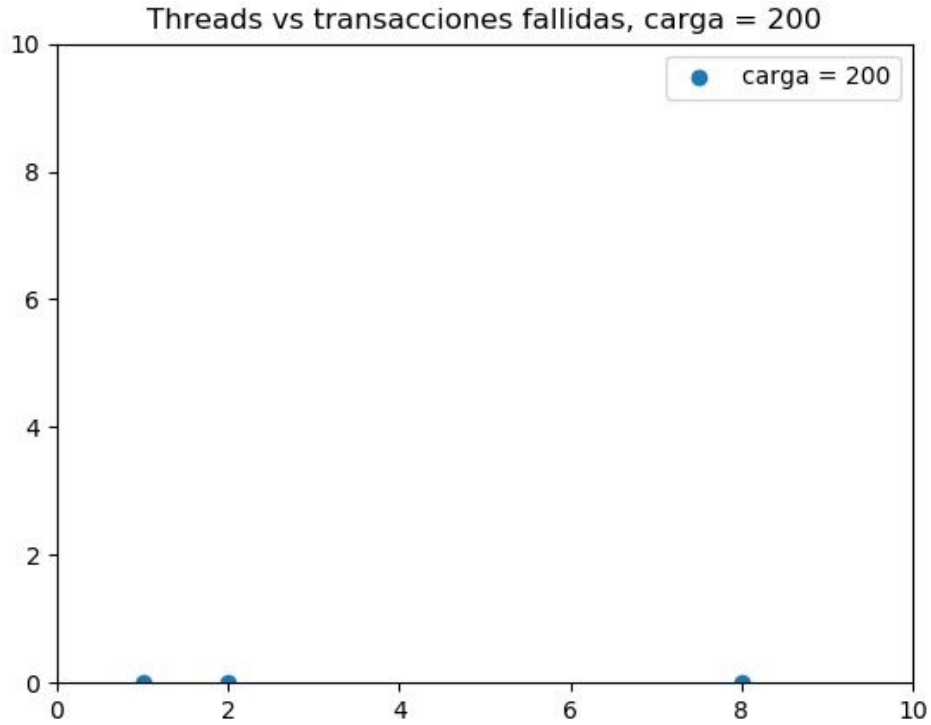
- **Número de transacciones perdidas:**

Carga 400, 20 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411

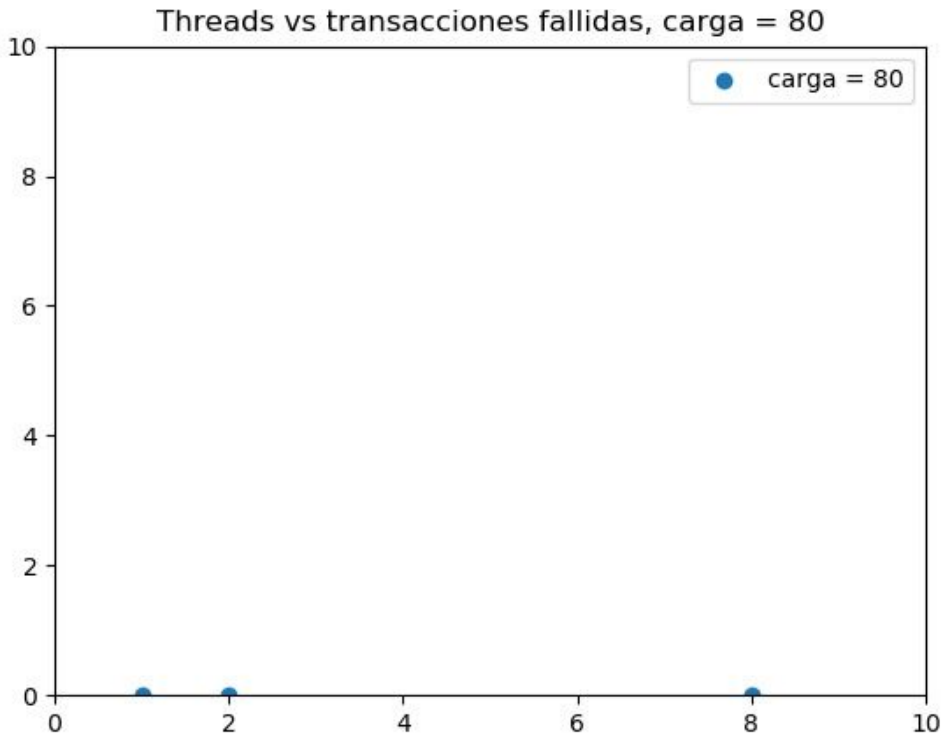


Carga 200, 40 ms de retardo:



Carga 80, 100 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411

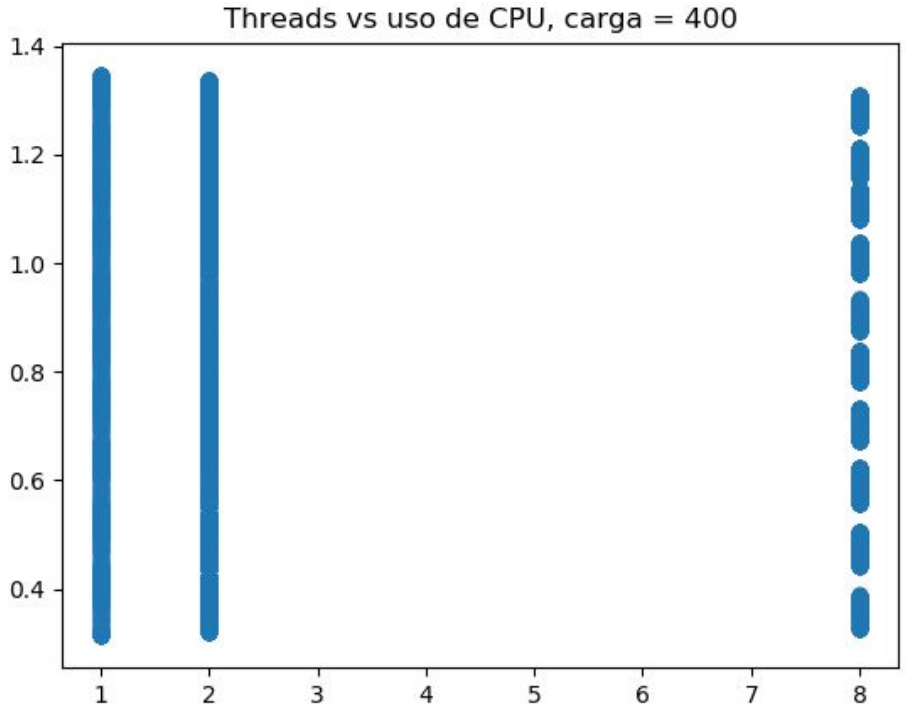


Se observó una mejora significativa en la cantidad de transacciones fallidas. No hubo. Esto tiene sentido pues ante una disminución en las operaciones computacionales, el servidor puede despachar de forma más rápida las tareas.

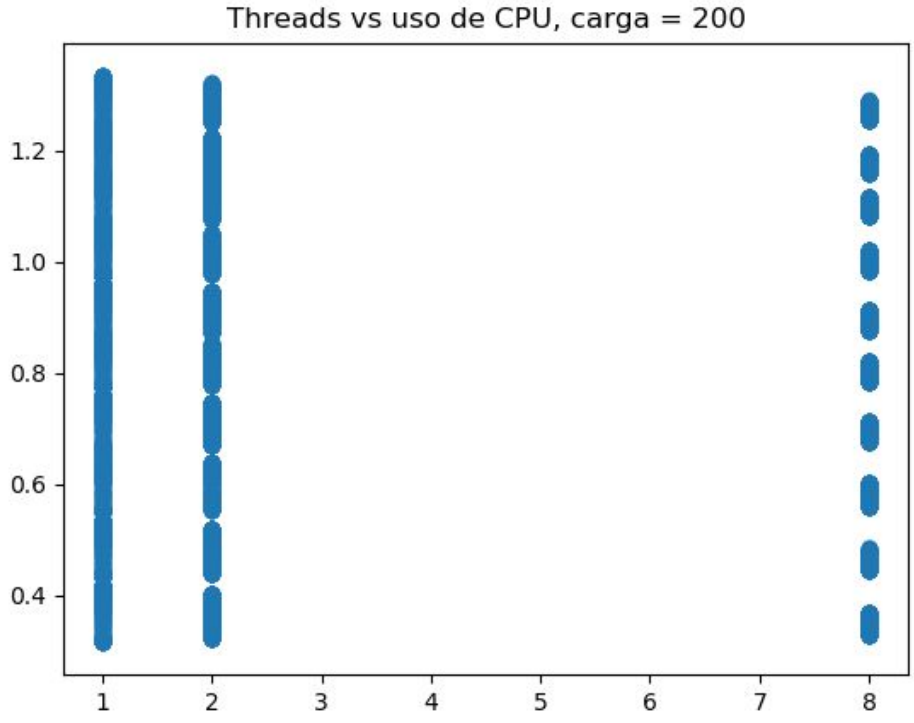
- **Porcentaje de uso de la CPU:**

Carga 400, 20 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411

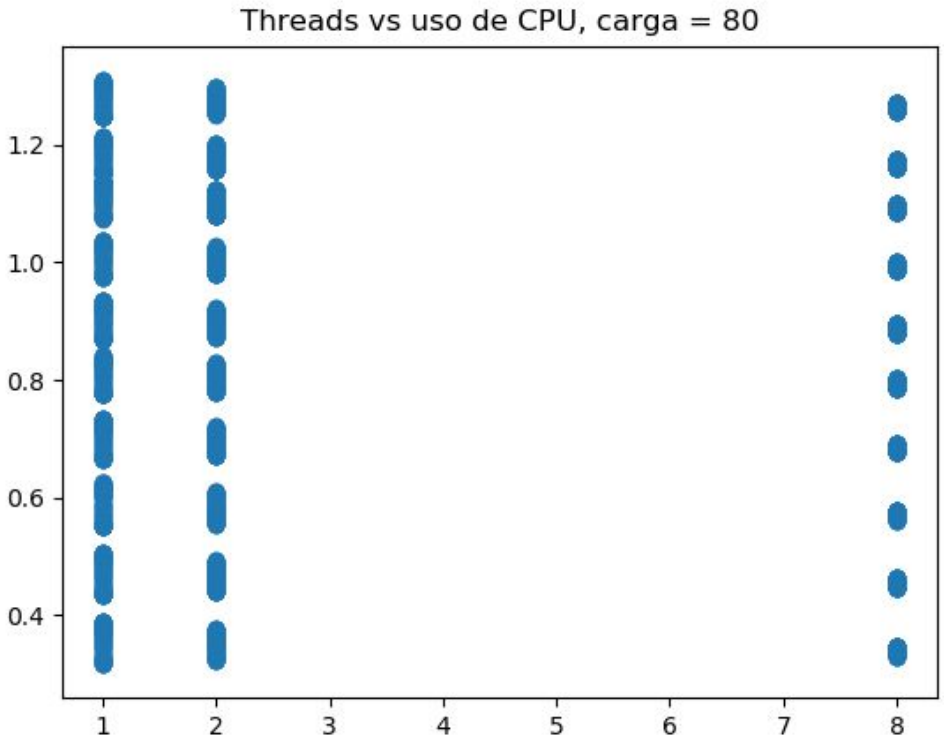


Carga 200, 40 ms de retardo:



Carga 80, 100 ms de retardo:

Mauricio Neira 201424001
Daniel Agudelo 201533411



Se observó el mismo patrón que en punto 3. No se vio una mejora en el uso de la CPU. Puede ser que el cambio en la cantidad de operaciones no fue el suficientemente significativo para disminuir su uso.