**Luis Alfonso Ruiz Botero - 201112453**

**Juan Daniel Carrillo Ramírez - 201613501**

**Infraestructura Computacional**

**Caso 3**

**1) Descripción de la Implementación**

Dado que se quería implementar ahora un pool de threads, se procedió a crear una constante con el número máximo de threads ejecutándose y se importaron las librerías ExecutorService y Executors.

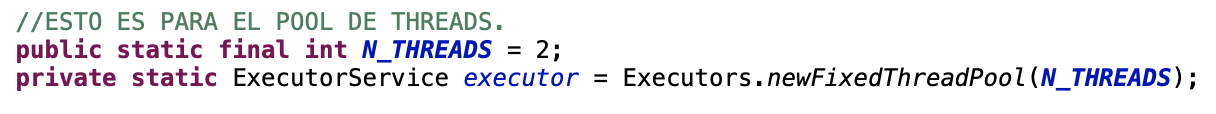


Imagen 1. Muestra la declaración para la inicialización del pool de threads.

Adicionalmente en el main, se eliminó la ejecución automática del thread creado, haciendo que fuera el ExecutorService quien manejara los threads.



Imagen 2. Muestra la declaración para la inicialización del pool de threads.

Del lado del cliente, se llevaron a cabo varias modificaciones para la implementación de los escenarios de carga. La primera, fue añadir timeStamps en los lugares donde era necesario calcular tiempos. Esto se llevó a cabo utilizando variables de tipo long con ayuda de System.CurrentTimeMillis() en los lugares donde se debían calcular los tiempos.

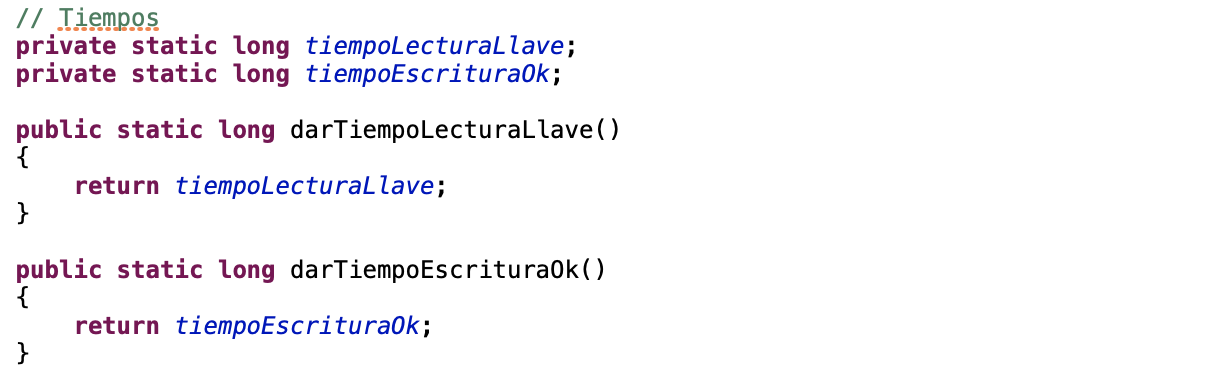


Imagen 3. Muestra la declaración para la inicialización de los tiempos.



Imagen 4. Muestra la declaración para la inicialización de los tiempos.

Adicionalmente, se realizaron cambios en la estructura de los protocolos. En la entrega pasada, era el usuario quien seleccionaba los algoritmos a utilizar, así como la información que mandaba con su ubicación. Debido a que se necesitaba simular usuarios concurrentes, estas selecciones se hicieron de manera automática (seleccionando siempre el primer elemento posible), y los mensajes con identificación y ubicación se enviaron con Strings por defecto.



Imagen 5. Muestra la declaración para la selección de algoritmos.

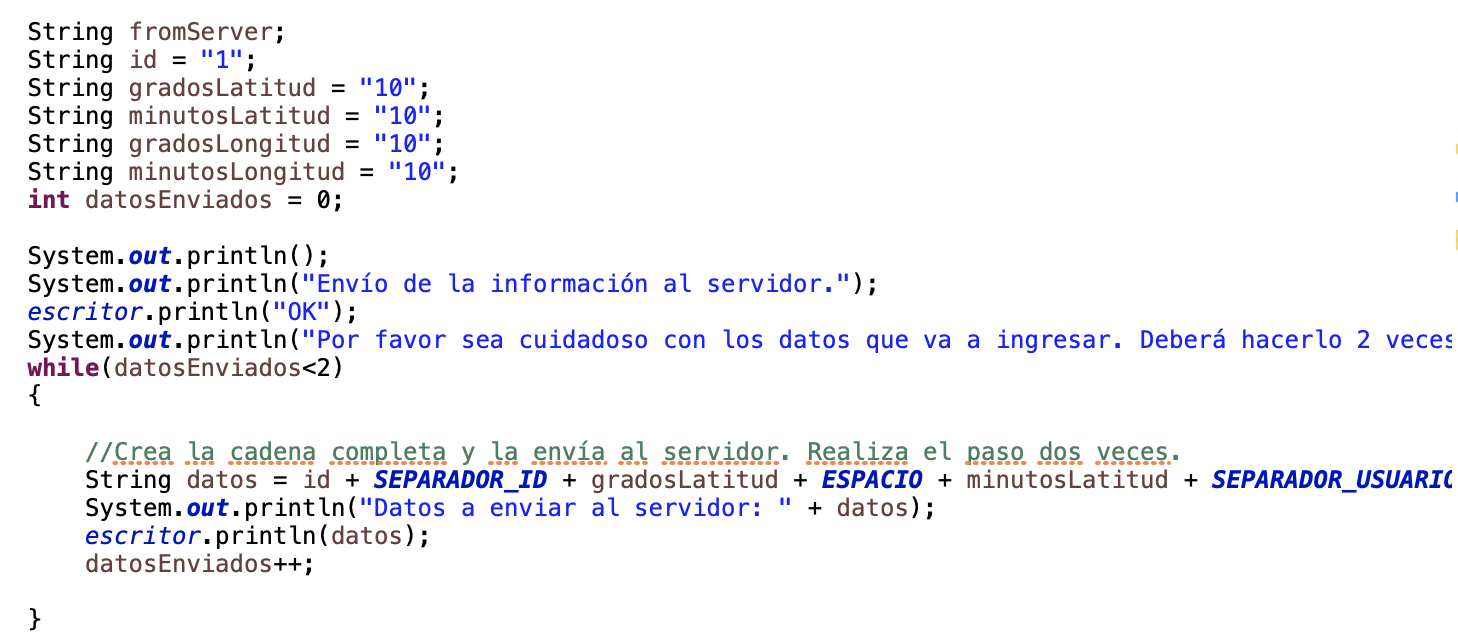


Imagen 6. Muestra la declaración para la selección de identificador y coordenadas.

Para terminar, se utilizaron las implementaciones de Gload y el método de getSystemCpuLoad() tal cual como estaban descritas en las guías.

**2) Identificación de la Plataforma**

Se realizó la identificación de la máquina donde se corrió el servidor. En un principio, se había dispuesto de 2 máquinas diferentes (una en donde se ejecutaba el servidor, y la otra donde se ejecutaba la carga), sin embargo, se presentaron algunos inconvenientes con dicha máquina, por lo que se debió ejecutar todo en la segunda. La configuración de la primera y segunda máquina se muestran a continuación:

**Máquina 1:**

**Arquitectura (32 o 64 bits):** 64 bits.

**Número de núcleos (cores):** 4 núcleos. 8 procesadores lógicos

**Velocidad del procesador:** 2.5 GHZ.

**Tamaño de la memoria RAM:** 12 GB.

**Espacio de Memoria asignado a JVM:** 5 Gb.

**Máquina 2:**

**Arquitectura (32 o 64 bits):** 64 bits.

**Número de núcleos (cores):** 2 núcleos. 4 procesadores lógicos

**Velocidad del procesador:** 2.9 GHZ.

**Tamaño de la memoria RAM:** 8 GB.

**Espacio de Memoria asignado a JVM:** 3 Gb.

En principio, la máquina 1 se encargaría del servidor. Sin embargo, después de realizar varias pruebas y obtener diferentes mensajes de error se decidió realizar todo en la segunda máquina.

Un error típico que se encontró era que, por ejemplo, el servidor no respondía de manera adecuada al envío de mensajes, como se puede observar en las siguientes imágenes:

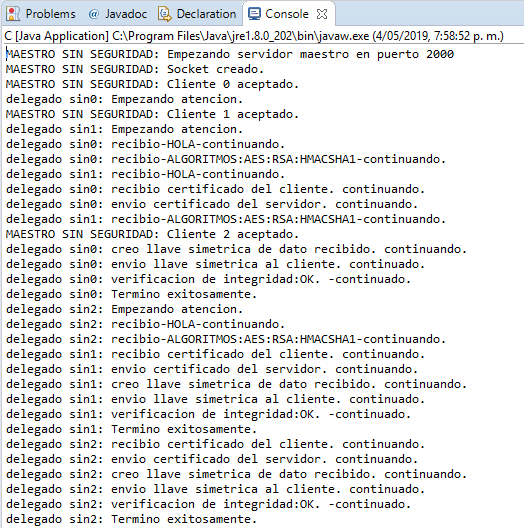


Imagen 7. El servidor funcionando correctamente al ejecutar 3 solicitudes diferentes.

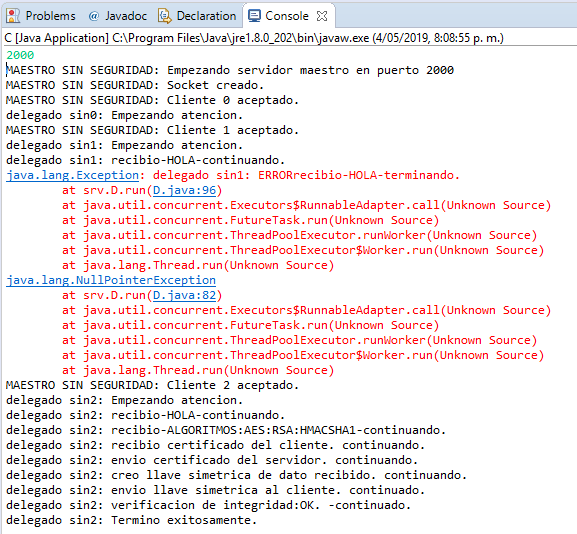
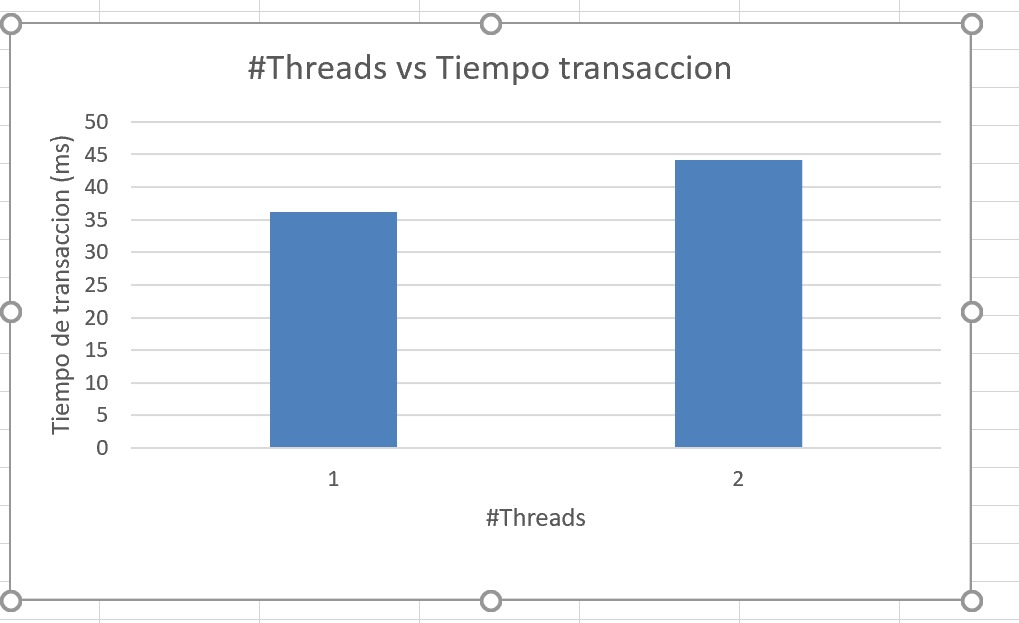
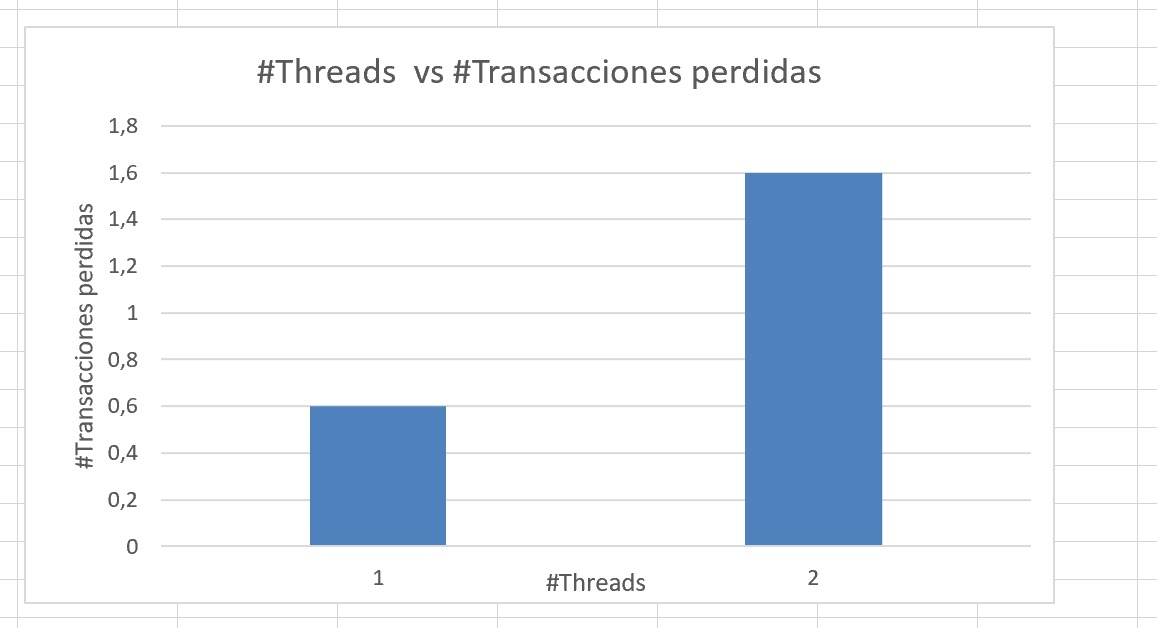


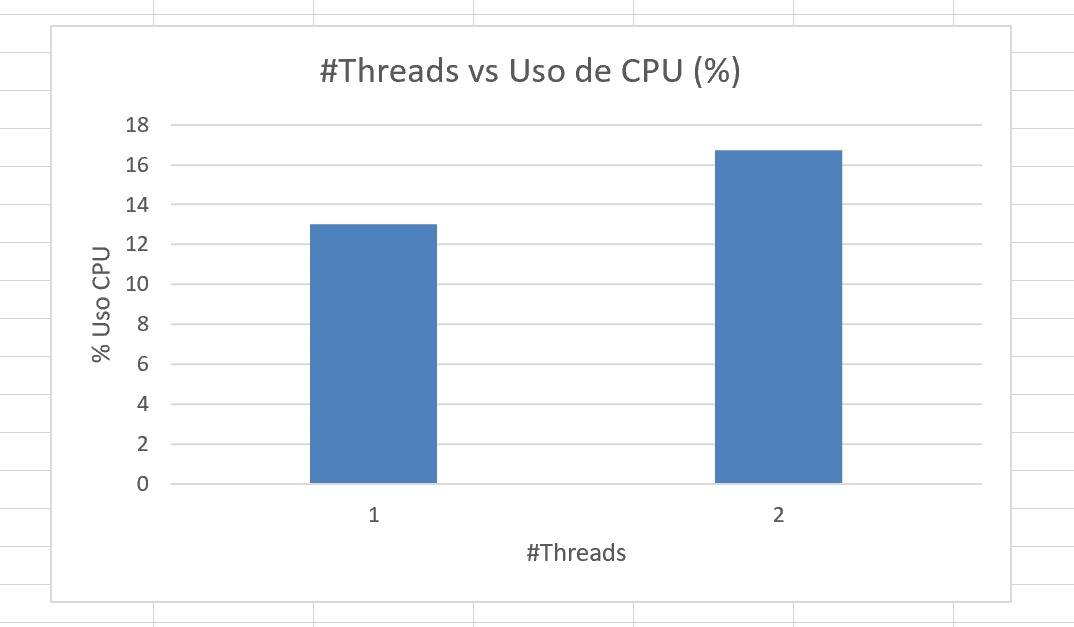
Imagen 7. El servidor funcionando de manera irregular, al ejecutar las mismas 3 solicitudes.

**3) Comportamiento de la aplicación con diferentes estructuras de administración de la concurrencia**

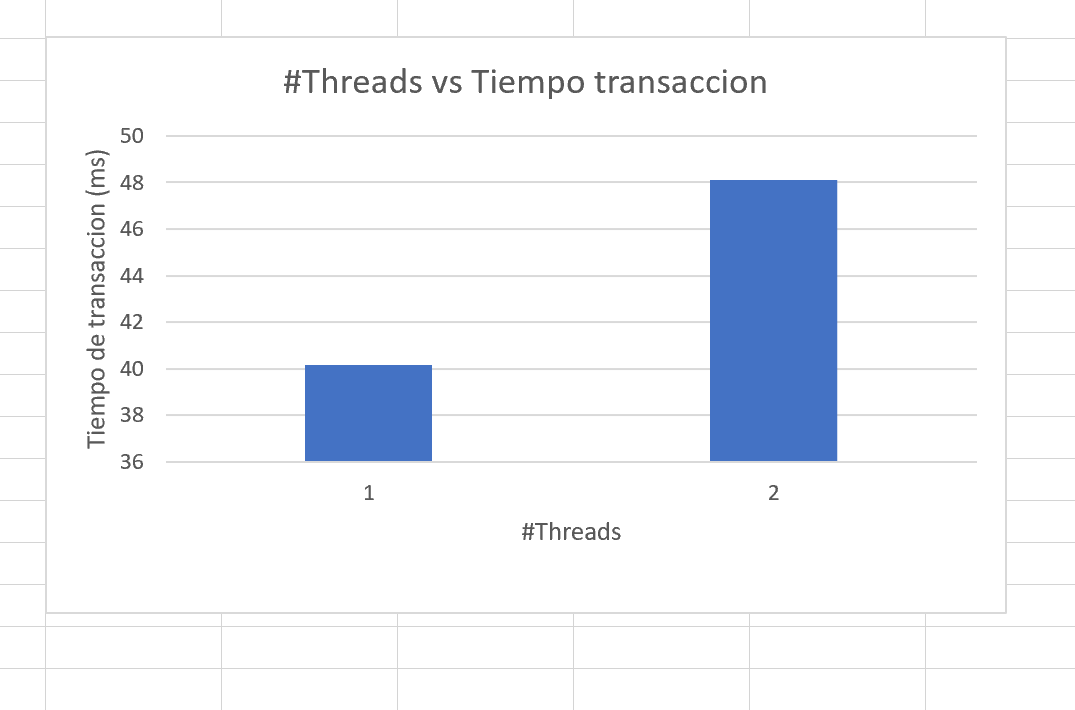
* 80 transacciones iniciadas con retardos de 100 ms.

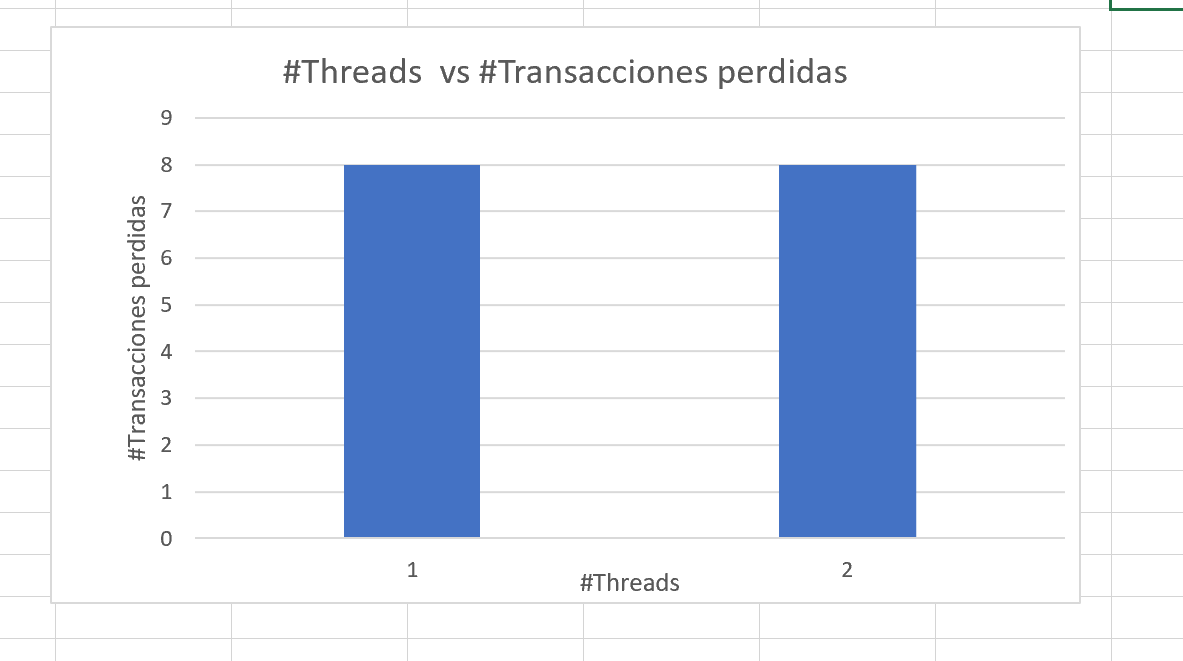


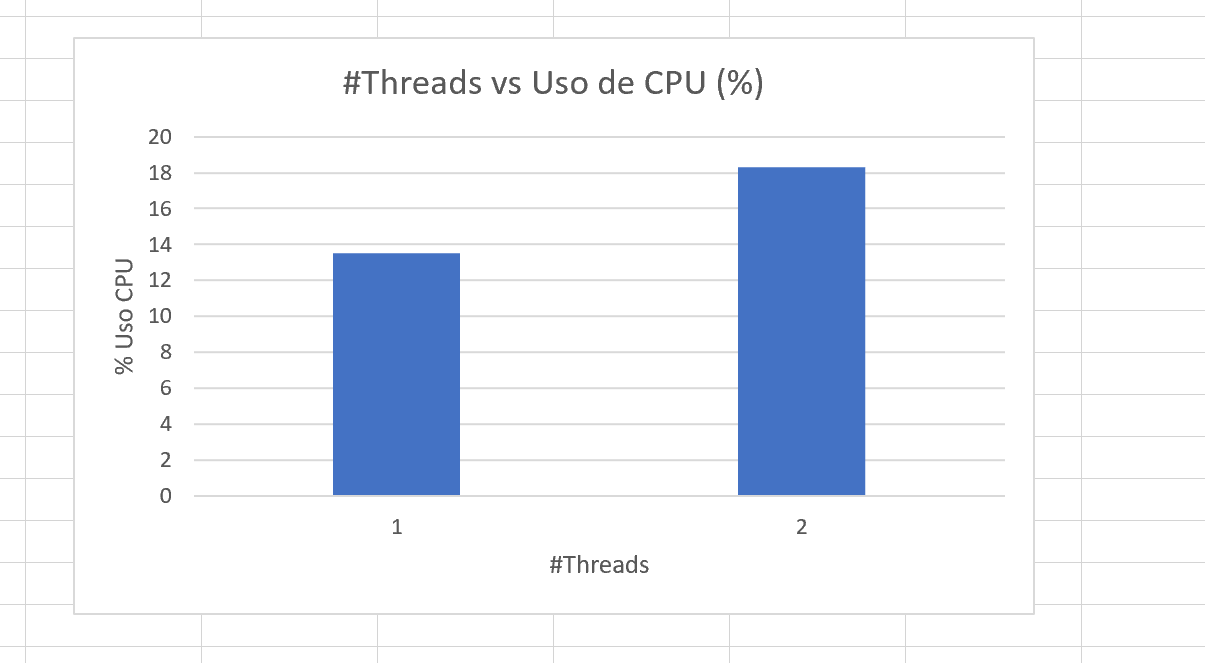




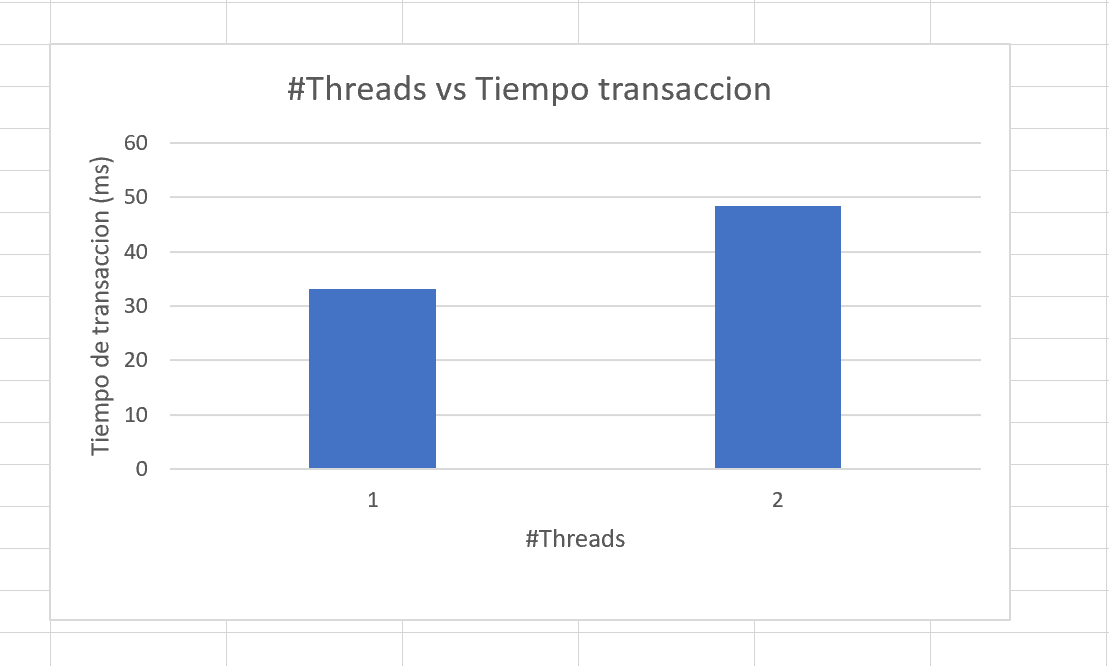
* 200 transacciones iniciadas con retardos de 40 ms

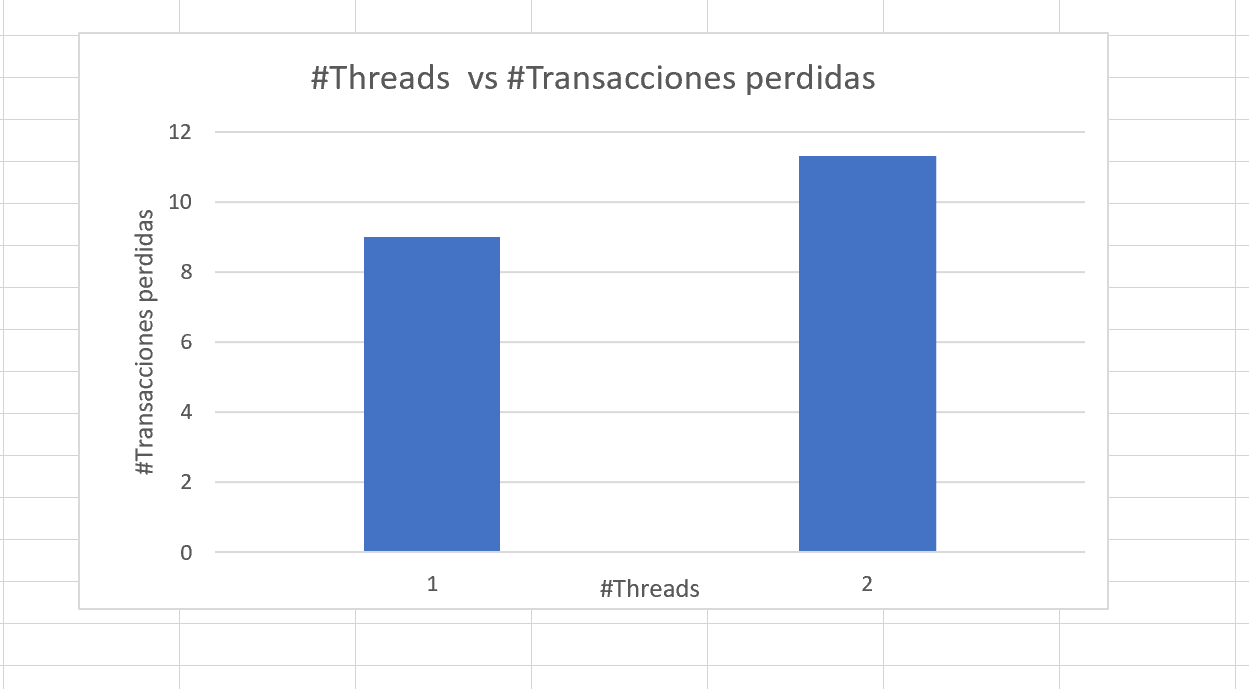


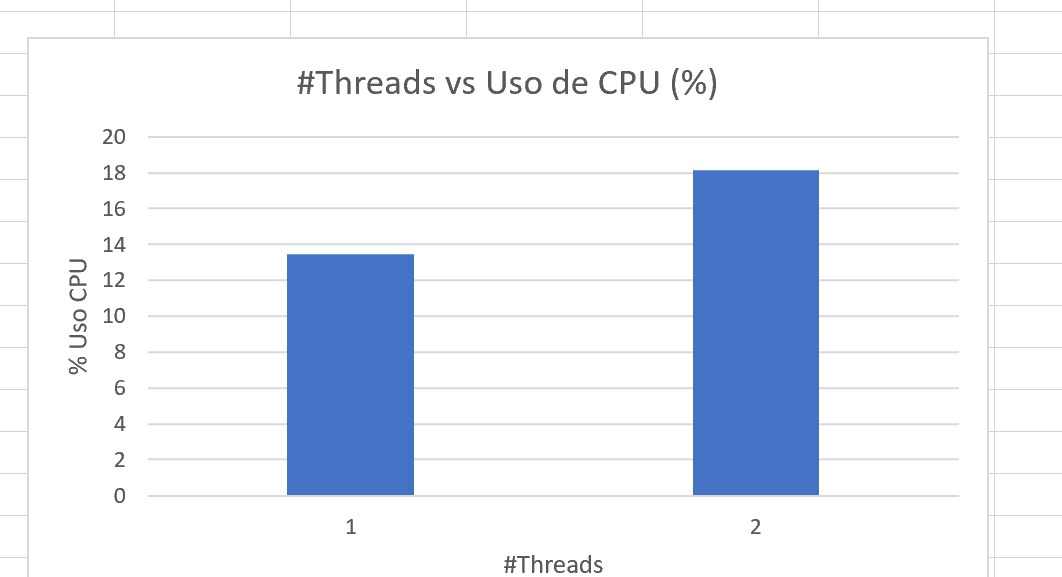




* 400 transacciones iniciadas con retardos de 20 ms







Se puede apreciar que las diferencias en tiempo promedio de transacción no son muy grandes. El tiempo que tarda el servidor desde que efectivamente atiende una solicitud y brinda una respuesta tiende a ser constante. Podría efectivamente existir una diferencia mucho mayor si se considerara el tiempo desde que se genera la solicitud y no desde que es atendida.

Las transacciones perdidas, del mismo modo, no presentan una variación significativa si se comparan en 1 o 2 threads. Sin embargo, a mayor número de transacciones, existe una mayor probabilidad de que se pierda alguna sin ser atendida.

Donde si se observa una variación considerable es en el uso de la CPU. Se observa cómo a mayor número de threads el uso de CPU aumenta también. La intensidad del uso de la CPU (uso en un momento de tiempo dado) no se ve afectada por el número de transacciones en total, sino por el número de threads atendiendo solicitudes.

Si tenemos en cuenta el tiempo de espera (aquel tiempo desde que una solicitud es creada y el tiempo donde es atendida) y el uso de CPU, encontramos que es una relación inversamente proporcional. A menor número de threads (menor uso de CPU), se tendrá que esperar más para ser atendido. A mayor número de threads (mayor uso de CPU) se tendrá que esperar menos para ser atendido. La clave está entonces en encontrar el punto de equilibrio dependiendo del negocio.

**2) Comportamiento de la aplicación ante diferentes niveles de seguridad**

No se realizaron pruebas para el servidor sin seguridad. Sin embargo, se esperaría que los tiempos de transacción fueran menores (debido a que no era necesario cifrar los datos de ubicación). Se esperaría igualmente que el uso de la CPU fuera un poco menor, pero siguiera manteniendo la relación de uso a medida que se aumentan los threads activos.