Infraestructura Computacional

Integrante 1: Lina Paola Cardozo Garzón **Código:** 201712455
Integrante 2: Andrés Ortiz Gómez **Código:** 201727662

Informe Caso 3

1. Descripción detallada de implementación de los monitores

A) Monitor de tiempo de respuesta

Este monitor se encarga de calcular el tiempo de ejecución de cada proceso. Comienza a contar el tiempo desde el momento en que se recibe el certificado del cliente y deja de contar después de que el servidor envía el registro de tiempo al agente. Una vez que finaliza la transacción, el programa almacena el tiempo transcurrido en un archivo llamado times.txt y continúa haciéndolo para el resto de tareas en el threadpool.

B) Monitor de % uso CPU

Para este monitor, creamos un hilo separado que se ejecuta mientras se ejecutan las tareas del threadpool. Para obtener el porcentaje de la CPU que se usa solo durante la ejecución de los procesos en el threadpool, necesitábamos iniciar y detener el hilo cpuMonitor en el momento adecuado. Entonces, hicimos que el hilo comience tan pronto el primer proceso ingrese a la ejecución del threadpool. Además, detenemos la ejecución del hilo cpuMonitor cuando todos los procesos en el threadpool han terminado de ejecutarse. Para determinar cuándo detener el hilo del monitor de CPU%, agregamos dos atributos estáticos a su clase: “tasksToProcess” y “processedTasks”. Cuando inicia el hilo cpuMonitor, el atributo “tasksToProcess” obtiene el número de transacciones por procesar que se ingresa por consola en el server-side. Durante la ejecución de un task, en cada momento que podría fallar la aplicación llamamos un método sincronizado en CpuMonitor que decrementa “tasksToProcess” por 1. Si el task logra terminar con exito, tambien se decrementa 1 de “tasksToProcess”, y además se incrementa por 1 “processedTasks” para contar cuantos tasks ya han terminado. Para parar la ejecución del monitor de cpu, simplemente miramos que “tasksToProcess” sea 0, lo cual significa que se acaban de ejecutar todos los tasks en el threadpool. Cada 100ms, el monitor manda el % de CPU usado en ese momento a un archivo llamado “cpu.txt”.

C) Monitor de Transacciones Perdidas

Usando los mismos atributos estáticos de la clase CpuMonitor podemos obtener el número de transacciones perdidas en la aplicación. Una transacción perdida la definimos como una transacción que por alguna razón, no alcanza a terminar del todo. Para obtener este número,

Integrante 1: Lina Paola Cardozo Garzón **Código:** 201712455
Integrante 2: Andrés Ortiz Gómez **Código:** 201727662

simplemente restamos el valor que tiene "processedTasks" del valor total de transacciones al momento de terminar la ejecución del cpuMonitor. Este valor lo mandamos a un archivo "lost.txt" que tiene el string "Lost Transactions: X" donde X representa el # e transacciones perdidas. Este valor se puede verificar de 2 formas:

i. Se puede inspeccionar el numero de lineas que hay en "times.txt" y restarlo del valor total de transacciones. Esto sirve porque se mide el tiempo de ejecución de cada task que se logra completar (se ignoran los valores de transacciones que no se completan).

ii. Se puede inspeccionar el numero de lineas que hay en "resultados.txt" y dividir este valor por 13 ya que cada transacción debe tener 13 líneas de log. Este valor se descuenta del valor total de transacciones.

Cambios al código:

A. Monitor de tiempo de respuesta
   a. D.java (servidor)
      i. En la línea 185 se inicia el monitor con "startTime"
      ii. En la línea 295 se termina el monitor con "endTime"
      iii. En la linea 310, se hace el llamado al método que guarda el valor del tiempo transcurrido al archivo "times.txt"

```
logTime(Long.toString(endTime - startTime));
```

      iv. En la línea 106, se declara el método usado para documentar los tiempos transcurridos(logTime).

B. Monitor de % uso CPU
   a. CpuMonitor.java (servidor)
      i. Se creó una clase nueva que extiende Thread para monitorear el uso de % CPU

C. Monitor de transacciones perdidas
   a. D.java (servidor)
      i. En las líneas 77,136,150,159,166,173,200,211,258, 262,274,303,312 se hace el llamado a `CpuMonitor.decreaseCountCpu();` . Estos son los puntos en los cuales se podría incrementar el número de procesos ya sea por que no se terminaron o porque se terminaron exitosamente.

D. Modificaciones al servidor C.java
   i. Se agrega lógica para pedirle al usuario el número de transacciones que se van a procesar en el threadpool.

Infraestructura Computacional

Integrante 1: Lina Paola Cardozo Garzón **Código:** 201712455
Integrante 2: Andrés Ortiz Gómez **Código:** 201727662

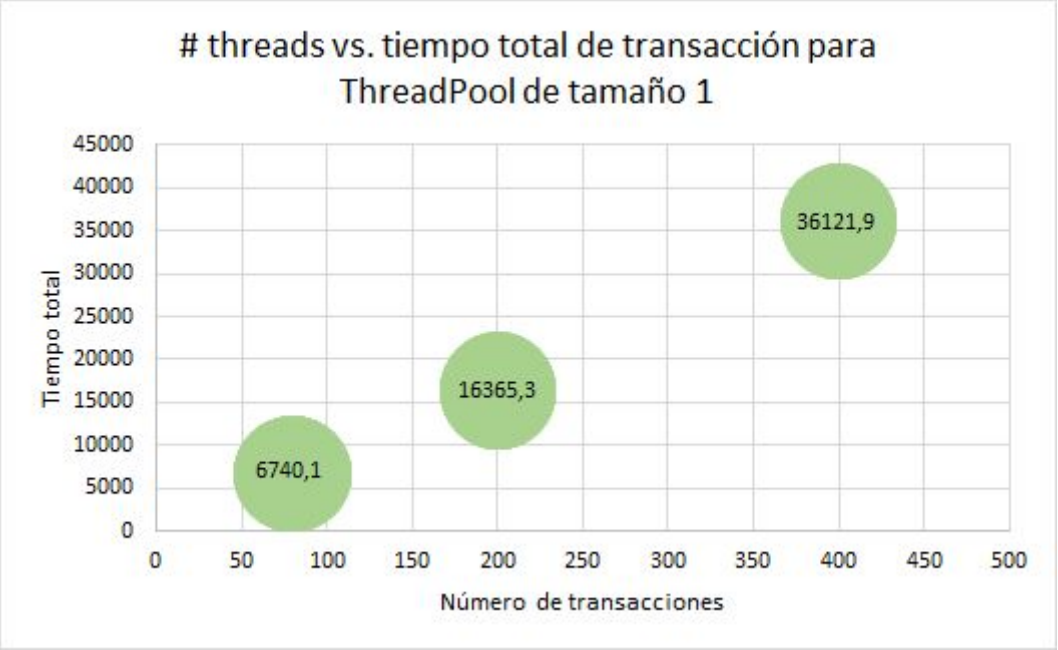
2. Identificación de la plataforma

   La máquina sobre la que corre el servidor es una máquina virtual con sistema operativo Windows 10.

   - Arquitectura: La máquina donde corre el servidor tiene una arquitectura de 64 bits.
   - Número de núcleos (cores): La máquina donde se corre el servidor tiene 2 procesadores principales.
   - Velocidad del procesador: Cada núcleo tiene una velocidad de 2,59GHz, es decir, que cada núcleo del procesador es capaz de realizar 2,59 mil millones de ciclos por segundo.
   - Tamaño de la memoria RAM: La máquina tiene una memoria RAM de 16GB.
   - El espacio de memoria inicial asignado a la JVM es de 128 MB y el espacio de memoria máxima que puede utilizar la JVM es de 2048 MB (2 GB). (Esto se observó en las configuraciones de IntelliJ IDEA 2020.2)

3. Respuestas punto 3

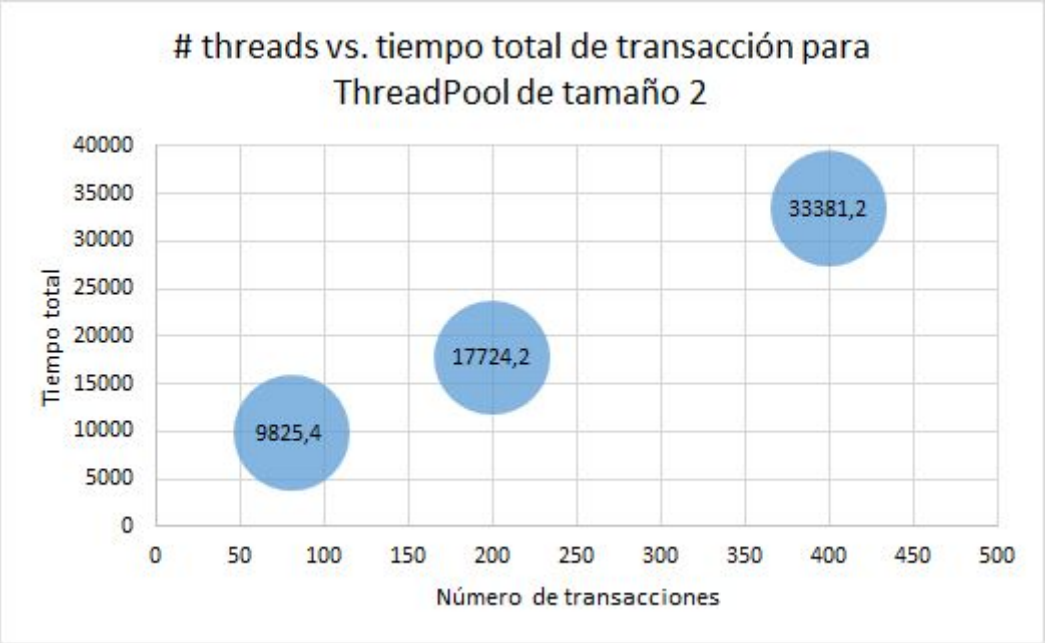
a. # threads vs. tiempo de transacción
   - Para ThreadPool de tamaño 1



   - Para ThreadPool de tamaño 2

Infraestructura Computacional

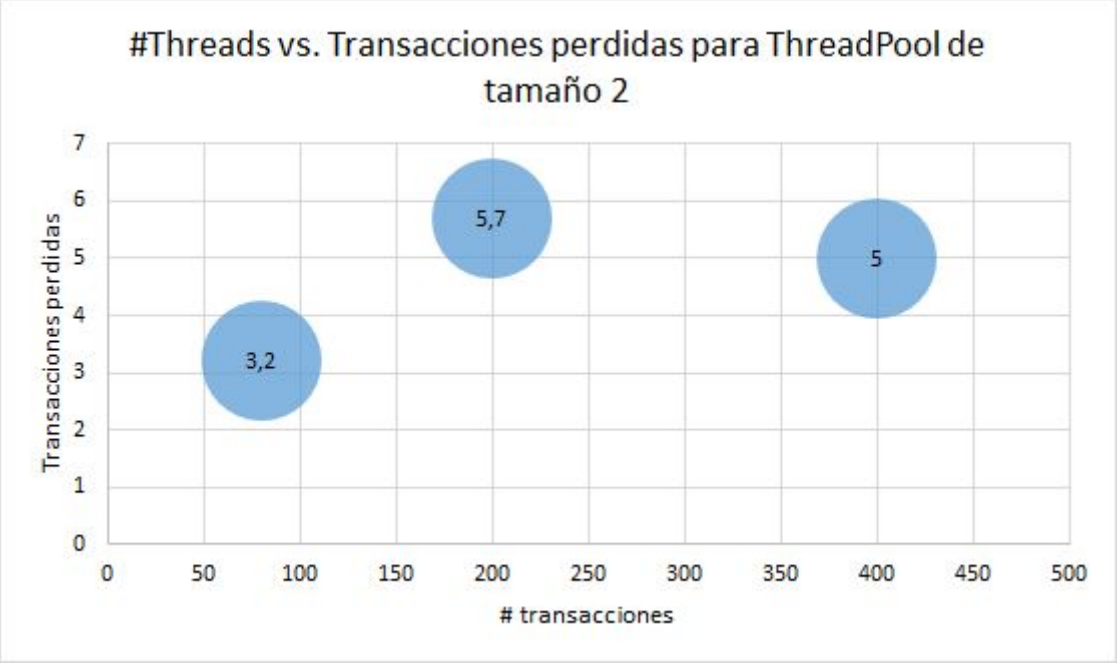
Integrante 1: Lina Paola Cardozo Garzón **Código:** 201712455
Integrante 2: Andrés Ortiz Gómez **Código:** 201727662



b. # threads vs. # de transacciones perdidas

- Para ThreadPool de tamaño 1: Para este caso no hubo transacciones perdidas, por lo que no hay gráfica.

- Para ThreadPool de tamaño 2

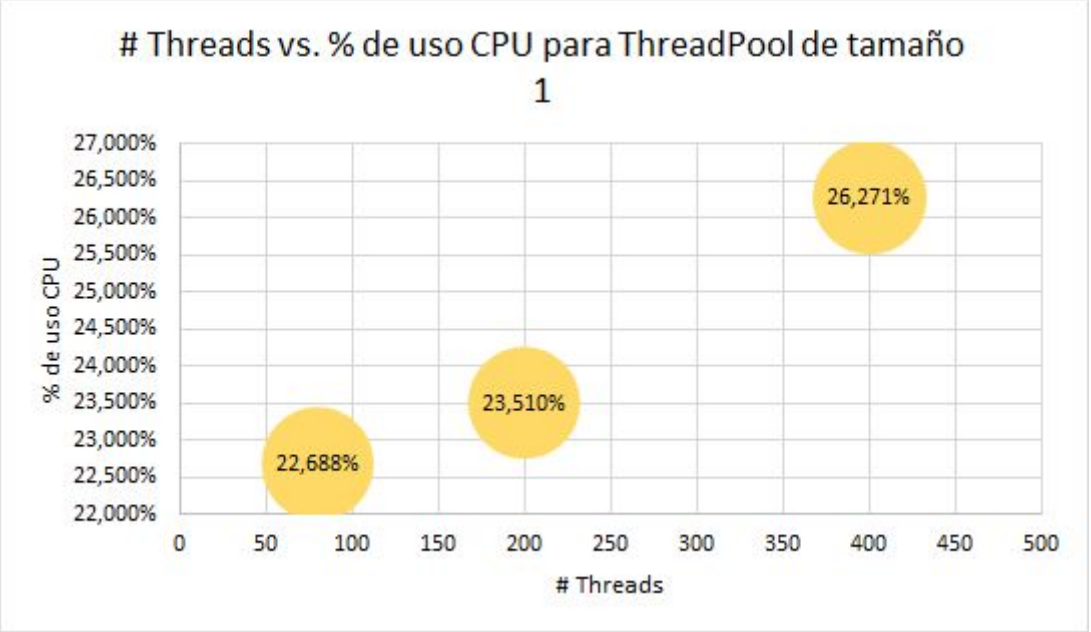


Infraestructura Computacional

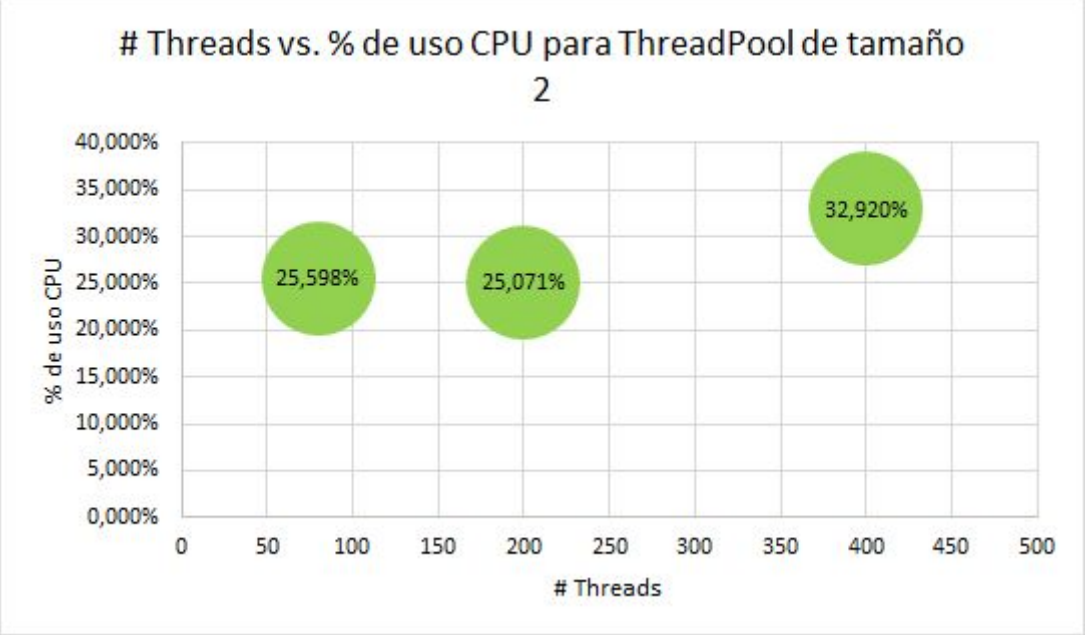
Integrante 1: Lina Paola Cardozo Garzón **Código:** 201712455
Integrante 2: Andrés Ortiz Gómez **Código:** 201727662

c. # threads vs. porcentaje de uso de la CPU

- Para ThreadPool de tamaño 1



- Para ThreadPool de tamaño 2



Análisis: Según los datos que hemos recopilado, el tiempo de respuesta para diferentes cargas en el conjunto de subprocesos aumenta linealmente a medida que aumenta el número de tareas en el conjunto de subprocesos. Esto se debe al hecho de

que a medida que aumenta el número de tareas, el tiempo total para manejar las tareas aumenta bien. Cuando tenemos sólo un proceso en el threadpool frente a dos, vemos pocos cambios en el tiempo de respuesta de la transacción para cargas bajas. Sin embargo, para cargas más grandes, notamos que el tiempo de respuesta difiere mucho en que el threadpool ejecuta las tareas más rápido con 2 threads que con uno. En cuanto a transacciones perdidas, con un thread en el threadpool, todas las tareas esperan su turno en la cola de threads esperando y reciben atención. Esto explica por qué no hay transacciones perdidas. Sin embargo, con un threadpool de tamaño 2, a veces las transiciones se pierden porque los subprocesos del servidor se retrasan al atender una tarea, y no alcanza atender bien el task que solicitó el servicio y esto a veces hace que el servidor omita las tareas del cliente. El consumo de % de CPU tiene a subir cuando se manejan más tasks en threadpool. Entre más tasks tenga que atender el servidor concurrentemente, más alto es el consumo de % de CPU. Si se usa un threadpool de 2 threads, el porcentaje de CPU sube, y esto se debe a que está usando más recursos del CPU, aprovechando los cores mientras tiene ventajas en otras partes.

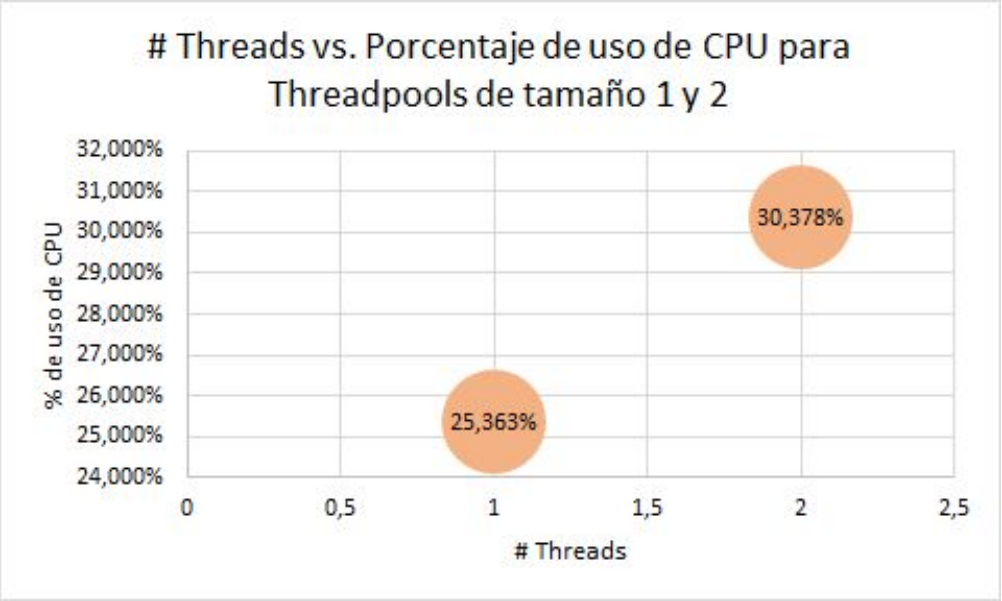
4. Respuestas punto 4

   Al comparar el comportamiento de una aplicación con seguridad y sin seguridad, uno esperaría que la aplicación sin seguridad tuviera un tiempo de respuesta más rápido ya que no se utiliza el tiempo para ejecutar los algoritmos de cifrado necesarios para garantizar una conexión segura. En las aplicaciones que tienen seguridad, lo que se pasa ejecutando el algoritmo introduce un retraso en el procedimiento de un protocolo de comunicación. Esto puede llevar a transacciones perdidas porque el thread en el threadpool no siempre podrá esperar a que se realice el cifrado y, por lo tanto, pasar a la siguiente transacción. Sin embargo, en una aplicación sin seguridad, los datos se transmiten directamente sin tiempos de espera para el cifrado. Entonces, uno esperaría que las transacciones se ejecuten más rápido y con menos pérdidas de transacciones. Además, como no se está calculando ningún cifrado, uno esperaría menor uso de la CPU.

   - # Threads vs. Porcentaje de uso de CPU

Infraestructura Computacional

Integrante 1: Lina Paola Cardozo Garzón **Código:** 201712455
Integrante 2: Andrés Ortiz Gómez **Código:** 201727662



Como se puede ver en la gráfica, los resultados concuerdan con la hipótesis ya que las transacciones usan menos % de CPU.