Autores:

Nicolás Francisco Abondano Velasco 201812467

Andrés Felipe Orozco González 201730058

# A Descripción monitores

* Monitores: La implementación de los monitores buscaba que se pudieran obtener datos del rendimiento del computador, pero específicamente del uso de CPU. Por esa razón el código es bastante simple y se utiliza la implementación dada en la guía para obtener la carga de la CPU. Dado que hay más de un Thread, el método es estático de tal manera que todos puedan acceder a él y obtener la información deseada. Es de notar que dado que suponemos que no hay perdidas en la comunicación entre el cliente – servidor, no se implementa en el monitor ningún método para llevar dichas estadísticas.
* Cliente: El código del cliente con seguridad es el mismo que el del proyecto pasado. Pero el código del cliente sin seguridad se modifico de tal manera que se cumpliera con el protocolo cuando no hay seguridad. Específicamente se hicieron cambios en el envió de mensajes, dado que ya no se utiliza la llave simétrica que se envía. Pero se conserva en el protocolo de cliente la rectificación del certificado del servidor y el no repudio con ayuda de funciones de hash para rectificar el mensaje enviado por el servidor. Esto ultimo se hace esta vez de manera diferente con ayuda de messageDigest.
* Servidor: El código del servidor se tiene que modificar en diferentes aspectos. Por un lado, se tiene que agregar a ambos servidores una variable en la clase D de tal manera que se pueda contabilizar el tiempo desde la fase 4 hasta la fase previa a que se termine la comunicación con el cliente. Este tiempo se agrega a un archivo que se ha creado previamente en constructor. Además de los cambios anteriormente mencionados, se realizan cambios adicionales en el servidor sin seguridad. En este se modifican las clases D y S. En S, se borran los siguientes métodos: ad, ae, hdg, sd, se y vi. Lo anterior se debe a que estos se utilizaban para hacer cifrado/descifrado simétrico/asimétrico. En D, se dejan las secciones de código que se encargaban de recibir y enviar las llaves de sesión que se negocian con el cliente. Pero la comunicación se da sin hacer uso de la llave y los algoritmos que se negocian. Sin embargo, para que la comunicación sea efectiva con el cliente se utiliza MessageDigest para enviar con una firma digital un mensaje al cliente.
* Generador: Generador se implementa de la misma manera para ambas pruebas y consiste en una clase que hace uso de loadGenerator para simular una carga de usuarios para el servidor, es ahí donde se ajustan los valores para llevar a cabo las pruebas.

# B Identificación de la plataforma

* Arquitectura: 64 bits, procesador x64
* Numero de núcleos: 2
* Velocidad del procesador: 2.4 GHz
* Tamaño de la memoria RAM: 4 GB
* Espacio de la memoria asignado al JVM:

# C Punto 3

Primero para verificar la calidad de todos los datos, decidimos en cada uno de los escenarios para cada transacción sacar un promedio, de forma que nos diera un ejemplo representativo de todo el escenario en uno solo. Además, a estos datos le sacamos la variación para cada transacción, de esta forma podíamos ver en una misma transacción que tanto variaba los datos según el intento. Sacamos el promedio de la variación en cada uno de los escenarios de cada uno de los valores (tiempo y % uso). Según se ve dentro del Excel en cada escenario la variación es muy pequeña comparada con el promedio en todos los casos, por lo que podemos considerar que los datos son de buena calidad. Hay un valor que no incluimos en las tablas, el cuál es el número de pérdidas, pues en ningún momento nos generó excepción ni faltaron transacciones por calcular, por lo que en todos los escenarios y experimentos el número de pérdidas fue 0.

La carga fijada para las gráficas fue de 400. Se realizaron 9 gráficas divididas en 3 “categorías”. La primera sería Tiempo de transacción vs # Threads, esta tiene 2 gráficas, una para 1 thread y otra para 2 threads, y una tercera gráfica que contiene ambas, de forma que se evidencie más sus diferencias.

Pero como estas gráficas tienen valores tan elevados en la primera transacción, para la siguiente categoría se llama igual, pues compara lo mismo, pero no incluye el primer valor. De esta forma pudimos evidenciar más en las gráficas el comportamiento del tiempo a través de las transacciones. (Se observa mejor en el excel)

En la otra categoría pusimos simplemente %Uso CPU vs # Threads, donde la primera gráfica era para 1 thread, la segunda para 2 threads, y la tercera contenía ambas.

Podemos evidenciar que al iniciar las transacciones siempre se tardaba mucho tiempo en la primera transacción, comparado con los demás tiempos. De forma contraria el % de uso del CPU en la primera transacción siempre salía como 0. Por las gráficas vemos que la variación entre 1 y 2 threads es principalmente en la cantidad, pues 2 threads tienden a gastar más tiempo por transacción y menos % de uso de CPU en cada transacción. Esto se debe a que 2 threads al atender más “clientes”, luego gastan más tiempo realizando las operaciones en total, que un solo thread. Pero la cantidad de CPU que gastan es menor ya que a pesar de atender más “clientes” son 2 threads trabajando conjuntamente, y el pedido que le hacen al procesador es menor.

Pero tanto 1 thread como 2 threads se comportan de manera muy similar, pues presentan casi siempre los mismos picos, o muy parecidos, es decir, según el número de transacción que se encuentran, su consumo de CPU o tiempo de transacción aumenta o disminuye de manera similar.

# D Punto 4

Como se hizo en el anterior caso, se hizo un promedio para cada uno de los escenarios de tal manera que se pudiera obtener un ejemplo representativo de todo el escenario en uno solo. Además, como ya se había hecho antes, se calculo la variación por cada transacción. De lo anterior obtuvimos resultados con los cuales podemos asegurar la calidad del escenario en el que solo se emplea un thread para responder las solicitudes, dado que la variación se mantiene bastante baja. Por otro lado, los datos que se generan para dos threads no tienen un patrón muy definido y la variación en algunos casos es mayor al 30%.

Lo que se esperaría confirmar es que el uso del CPU sea mucho más bajo cuando la aplicación no tiene que cifrar para comunicarse, dado que las operaciones de cifrado, en especial las asimétricas tienen un alto costo de computación. Además, también se esperaría que dos threads generen un mayor uso de la CPU.

Como se ve comparando las gráficas, lo que obtuvimos no esta totalmente de acuerdo con lo que esperábamos, especialmente porque para dos threads parece ser que una aplicación de comunicación sin seguridad utiliza un mayor porcentaje de CPU que una aplicación con seguridad. No obstante, esto se puede explicar a partir de la alta variación que se obtuvo en la toma de dichos datos. Es de destacar que en la toma de datos se trato de mantener al máximo el mismo ambiente de pruebas para evitar distorsiones.