Caso 3

Fabio Andrés López Corredor 201423782
Margarita Gómez Ballén 201423591
Jose Gabriel Tamura Lara 201424484

Preparación

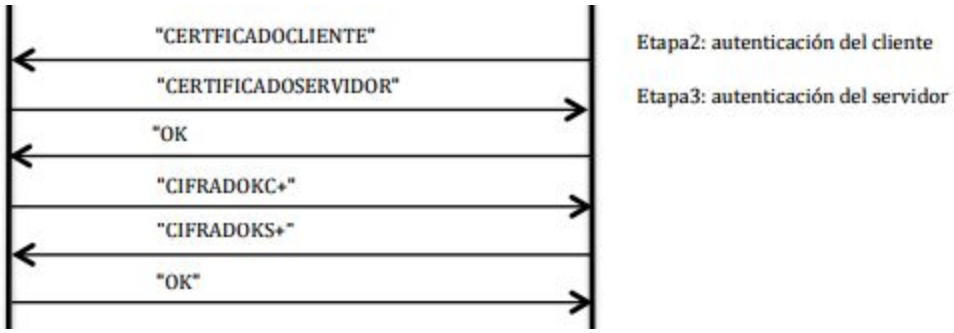
Tiempo de autenticación

Cliente sin seguridad

Para medir los lapsos de tiempo que se pedían en este caso se hizo uso del método

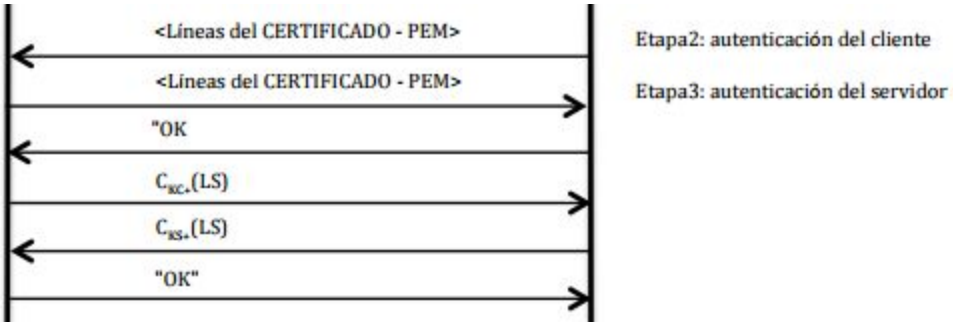
```
System.nanoTime();
```

El cual devuelve el valor actual de la fuente de tiempo de la Máquina Virtual de Java en nanosegundos.
De esta manera, se dio inicio para contar este tiempo justo antes de que el cliente mandara su certificado y acaba cuando este recibe el “OK” después de haber mandado “CIFRADOKS+”.
Intercambio de información que se da durante dicho lapso de tiempo:



Cliente con seguridad

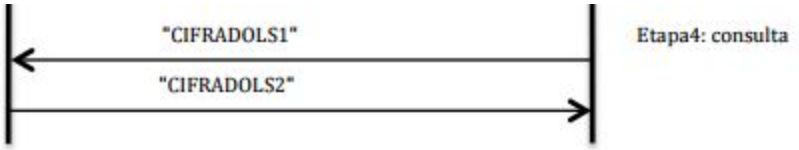
En este caso se dio inicio también antes de que se envíe el certificado y acaba cuando recibe el “OK” que resulta de haber mandado el cifrado con la llave pública del servidor.
Intercambio de información que se da durante dicho lapso de tiempo:



Tiempo de consulta

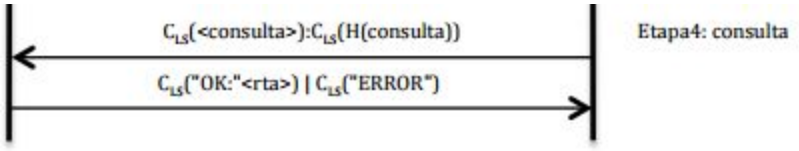
Cliente sin seguridad

Se dio inicio a la medida del tiempo antes de que el cliente mande “CIFRADOLS1” y termina luego de que el servidor responda con “CIFRADOLS2”.
Intercambio de información que se da durante dicho lapso de tiempo:



Cliente con seguridad

Se inicia a contabilizar este tiempo antes de que el cliente mande la consulta cifrada con la llave simétrica anexada con el hash de la consulta y termina una vez que el se reciba como respuesta el “OK” cifrado por parte del servidor.
Intercambio de información que se da durante dicho lapso de tiempo:



- Cada cliente que fuera creado e iniciara una conversación con el cliente tendría como variables estos dos tiempos. Cuando acabara, el thread escribe en un archivo estos dos valores en milisegundos para que quede registrado y pueda ser promediado después.

```
try{
    File tiempos = new File("./data/tiempos");
    PrintWriter writer = new PrintWriter(new FileWriter(tiempos,true));
    writer.println("tAutenticación:"+ TimeUnit.MILLISECONDS.toMillis(tiempoAutenticacion));
    writer.println("tRespuesta:"+ TimeUnit.MILLISECONDS.toMillis(tiempoRespuesta));
    writer.close();
    }catch (Exception e){
        e.printStackTrace();
    }
```

Número de transacciones perdidas

Cliente sin seguridad

En caso de que se presente una excepción en cualquier momento de la conversación, se tomaba la transacción como “perdida”, de esta manera el método de iniciarConversacion() se

hacía dentro de un try-catch, donde se imprimía una línea con “falla” en el archivo de texto para indicar esta pérdida.

```
try
{
    canal = new Socket(DIRECCION, PUERTO);
    out = new PrintWriter(canal.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(canal.getInputStream()));

    iniciarConversacion();
}
catch (Exception e)
{
    e.printStackTrace();

    try{
        File tiempos = new File("./data/tiempos");
        PrintWriter writer = new PrintWriter(new FileWriter(tiempos,true));
        writer.println("falla:m");
        writer.close();
        }catch (Exception x){
            //:)
        }
}
```

Cliente con seguridad

Para este cliente se realiza lo mismo que en el anterior, para así tener constancia de que hubo una falla en el archivo de texto.

```
public void iniciarComunicacion()
{
    try
    {
        canal = new Socket(DIRECCION, PUERTO);
        out = new PrintWriter(canal.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(canal.getInputStream()));

        iniciarConversacion();
    }
    catch (Exception e)
    {
        try{
            File tiempos = new File("./data/tiempos");
            PrintWriter writer = new PrintWriter(new FileWriter(tiempos,true));
            writer.println("falla:m");
            writer.close();
            }catch (Exception x){
                x.printStackTrace();
            }
    }
}
```

Porcentaje de uso de la CPU

Para este indicador se hizo uso de la herramienta de Windows de Monitor de Rendimiento para ambos casos. Se tuvo en cuenta el contador de porcentaje de tiempo de procesador para el proceso que realizaba el programa en ejecución (javaw).

Identificación de la plataforma

La máquina utilizada para correr el servidor cuenta con las siguientes especificaciones:

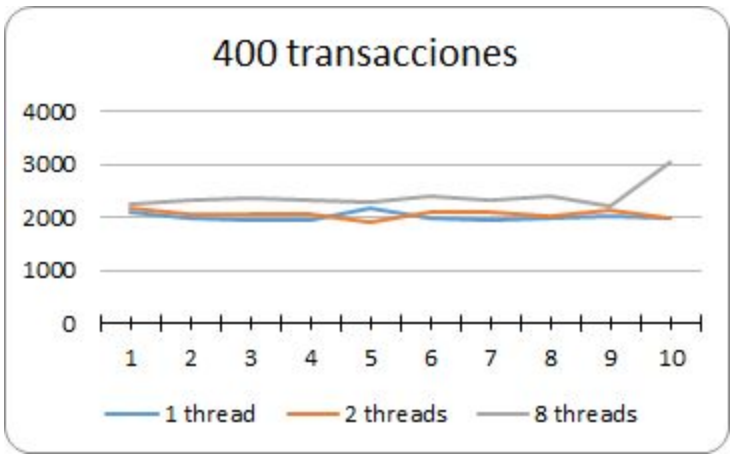
- Arquitectura de 64 bits
- Núcleos: 2
- Procesadores lógicos: 4
- Velocidad del procesador: 2.00 - 2.60 GHz
- Tamaño de la memoria RAM: 8 GB
- Espacio de la memoria asignada a la JVM: 1700 MB

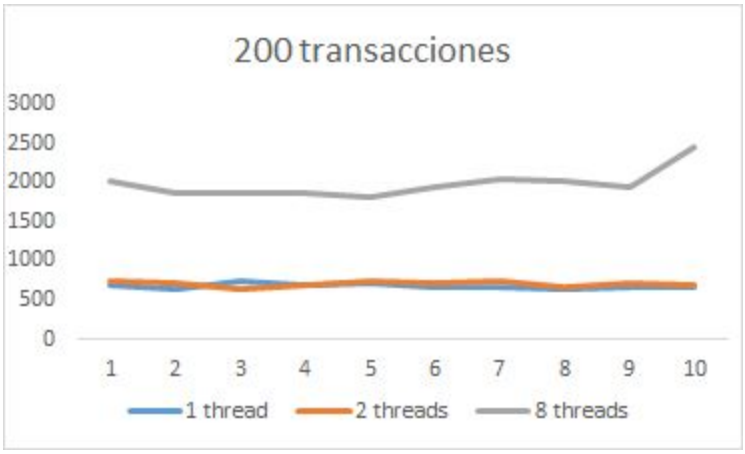
Comportamiento de la aplicación con diferentes estructuras de administración de la concurrencia

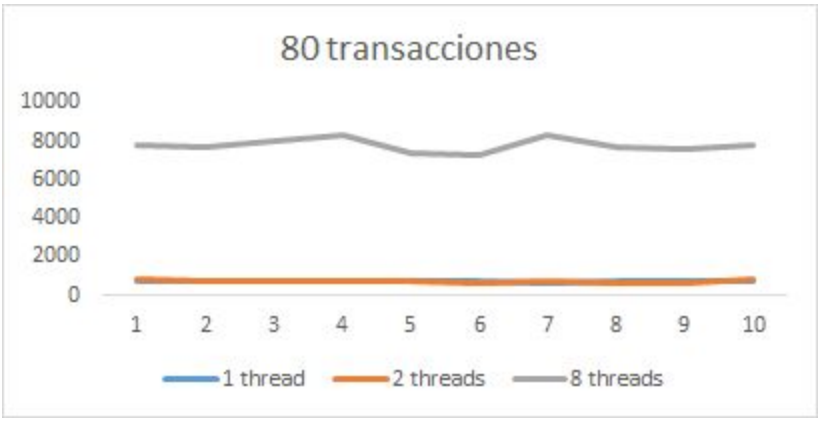
Gráficas

Número de threads vs. tiempo de autenticación

Eje x: # prueba
Eje y: tiempo de autenticación en milisegundos

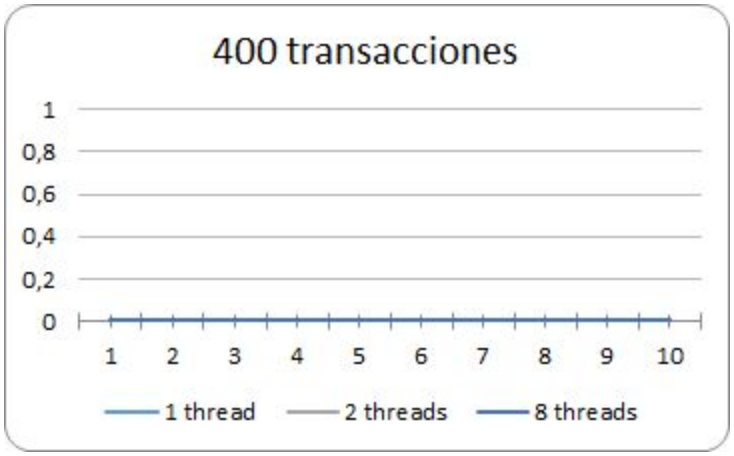


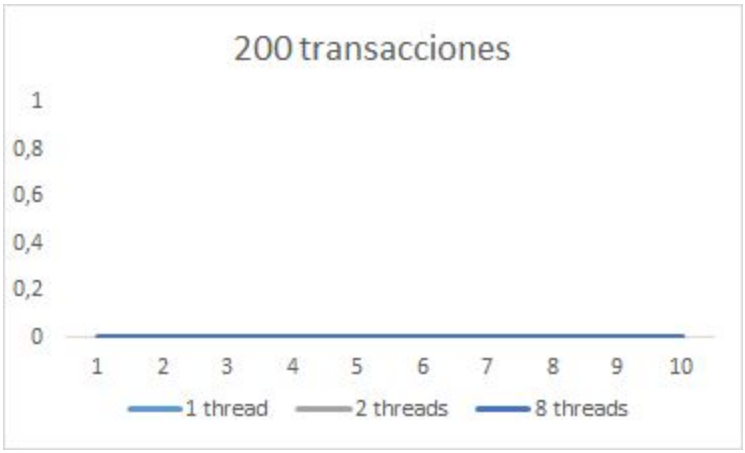


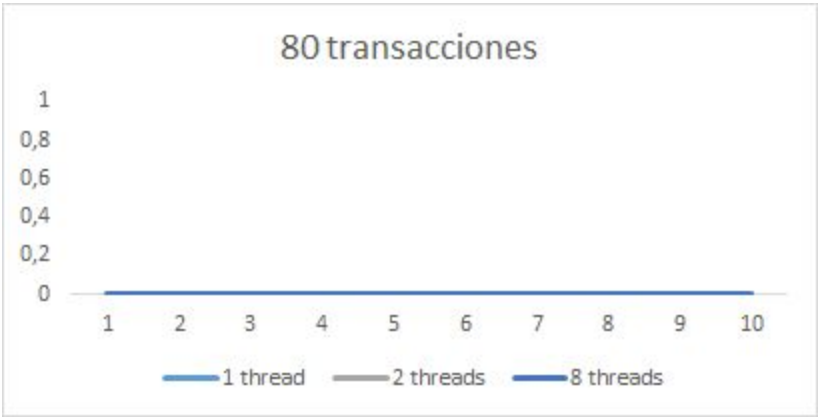


Número de threads vs. número de transacciones perdidas

Eje x: # de prueba
Eje y: # de transacciones perdidas

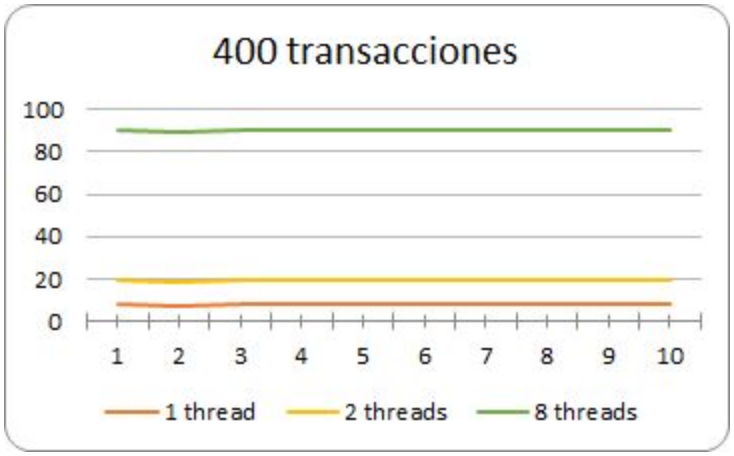


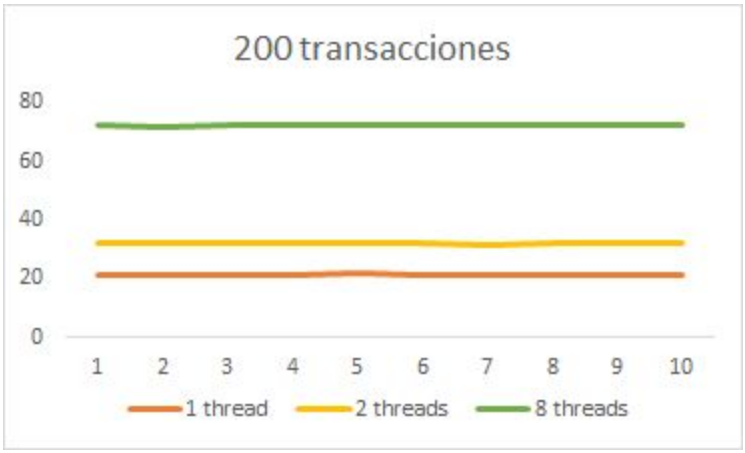


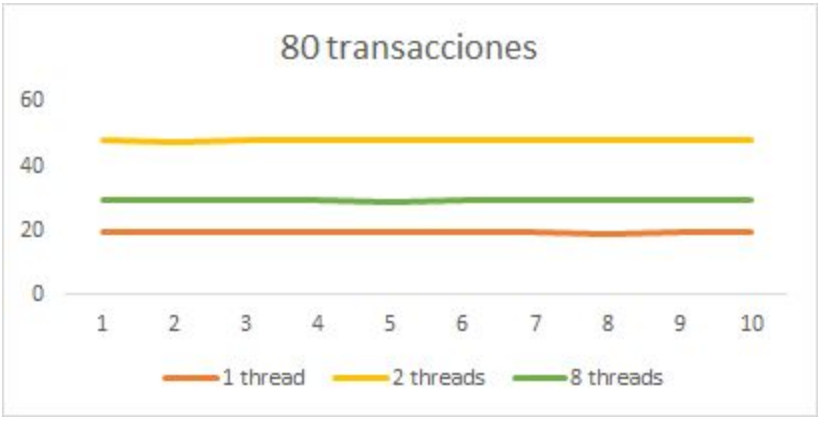


Número de threads vs. porcentaje de uso de la CPU

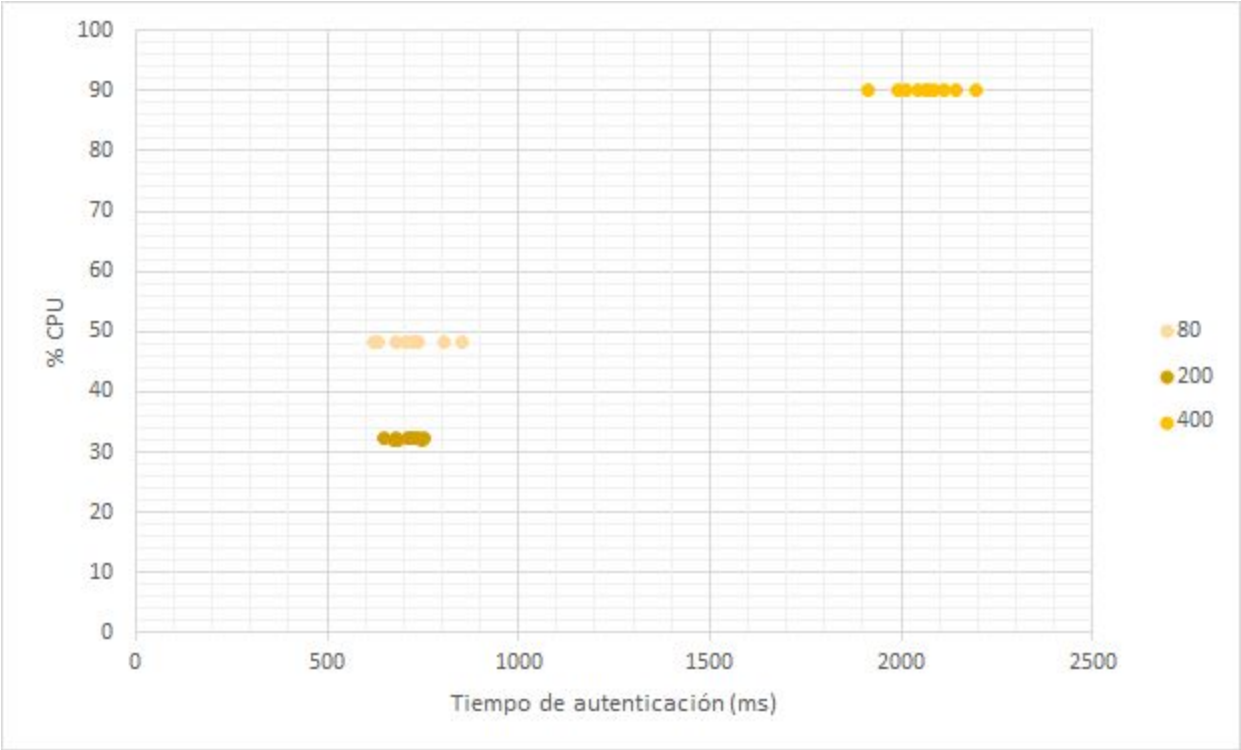
Eje x: # de prueba
Eje y: % CPU


400 transacciones
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
1 thread
2 threads
8 threads

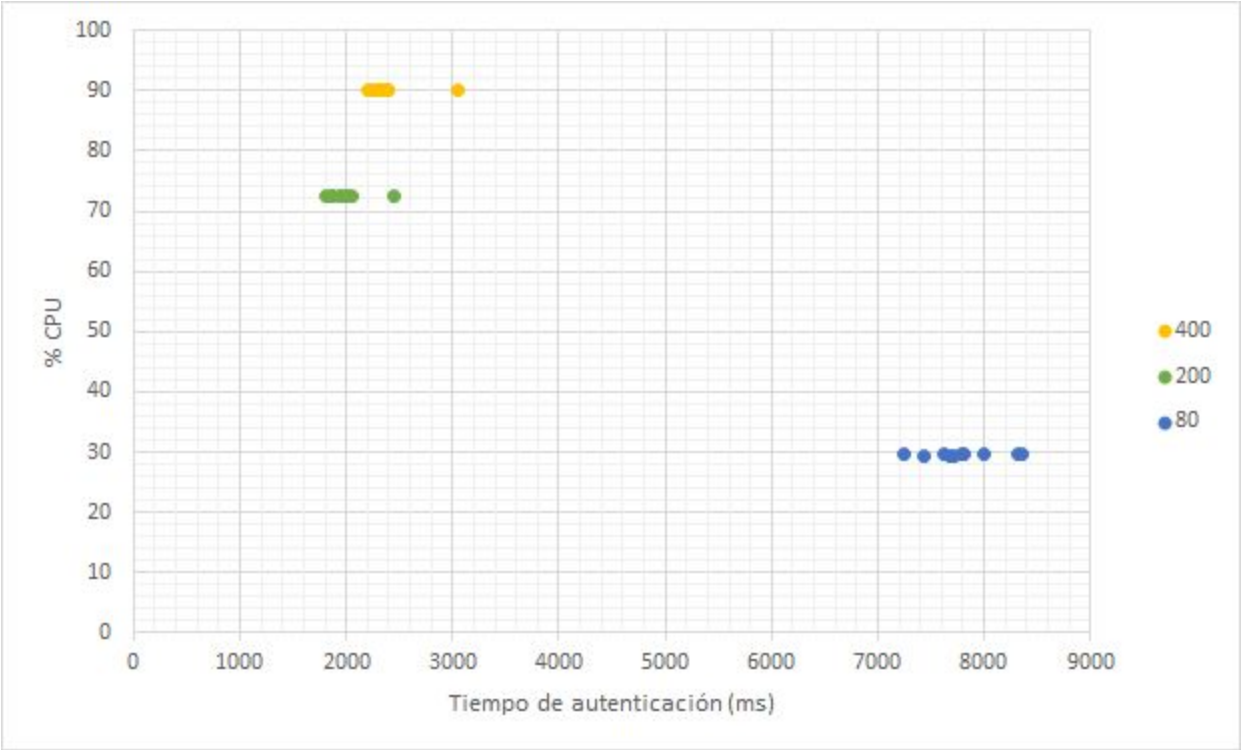

200 transacciones
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
1 thread
2 threads
8 threads


80 transacciones
60
40
20
0
1 2 3 4 5 6 7 8 9 10
1 thread
2 threads
8 threads

Tiempo de autenticación vs. porcentaje de uso de la CPU con 2 threads



Tiempo de autenticación vs. porcentaje de uso de la CPU con 8 threads



Conclusiones

Como se puede ver en las gráficas, hay una estrecha relación entre las variables que se controlan y el rendimiento de la máquina donde se ejecutan las pruebas. Intuitivamente esta relación mostraría una relación inversamente proporcional entre, por ejemplo, el número de threads y el tiempo de autenticación. Sin embargo en la práctica se ve cómo diferentes escenarios pueden hacer que una usualmente deseable configuración se vuelva realmente ineficiente.

Al comparar el número de threads vs. el tiempo de autenticación se vió que al aumentar el número de threads también aumentaba el correspondiente tiempo (contrario a lo que se esperaba). Especialmente, la brecha entre el sistema con 8 threads contra los otros dos se hacía más grande al haber menos transacciones por atender. Analizamos este fenómeno y concluimos que la baja cantidad de transacciones hace que la ejecución concurrente se vuelva más un peso que una ayuda. Esto quiere decir que la administración de muchos threads termina tomando el tiempo que una ejecución secuencial resolvería de manera eficiente. Se propone hacer pruebas con más transacciones, donde el comportamiento del pool permitiría resultados más cercanos a lo inicialmente esperado. Para esta serie de pruebas también se encontró un comportamiento más variado en cada una de las 10 repeticiones a medida que se aumentaba el número de threads.

Para este caso no hubo transacciones perdidas. Esto ocurre gracias a que el espacio de memoria asignado a la Java Virtual Machine, junto al equipo utilizado para realizar las pruebas,

contaron con suficiente espacio y capacidad para dar respuesta a todas las solicitudes de los clientes. Esta situación tiene sus efectos en el desempeño del sistema, pues al operar con estos volúmenes de datos percibimos un aumento en la latencia y en los índices de rendimiento, que mostraban que el procesador estaba llegando al límite de su capacidad.
Al comparar porcentaje de uso de CPU con el número de threads se ve como un incremento en el pool genera también un incremento en el porcentaje de uso. A pesar de que este es un resultado acorde a lo esperado, es de notar que en el caso en que hay 80 transacciones la configuración con 2 threads es la que tiene mayor porcentaje de uso de la CPU. Teniendo en mente esto y los resultados de la primera serie de gráficas se concluye que en este mínimo número de transacciones 2 threads requieren un mayor trabajo para terminar realizando el desempeño de uno solo.
Fijando 2 threads se observan puntos que reflejan por cada tiempo de autenticación el porcentaje de uso de la CPU que utilizó el sistema. Estas tablas son diagramas de dispersión para facilitar su lectura.Dicho esto, se puede ver como el tiempo de autenticación tiene un comportamiento flexible mientras que el porcentaje de uso de la CPU es casi constante. Se esperaba que a mayor cantidad de threads menor tiempo de autenticación, lo cual no siempre se da (no se nota una gran diferencia entre 400 threads y 200 threads pero si fueron tiempos mucho mejores a los de 1 solo thread). Esta situación se le atribuye al método que se usa para medir la última variable y el tiempo reducido entre las repeticiones del experimento que se realizan.
Al realizar la prueba equivalente para el caso de 8 threads se obtiene el mismo resultado en cuanto a la relación entre el porcentaje de uso de la CPU y el tiempo de autenticación. Es de notar cómo el tiempo de autenticación se eleva al tener tan solo 80 transacciones. El tiempo disminuye para 200 transacciones y aumenta levemente para 400 transacciones. Es el comportamiento esperado ya que esta cantidad de threads se vuelve funcional con largas filas de operaciones a realizar.

Comportamiento de la aplicación ante diferentes niveles de seguridad

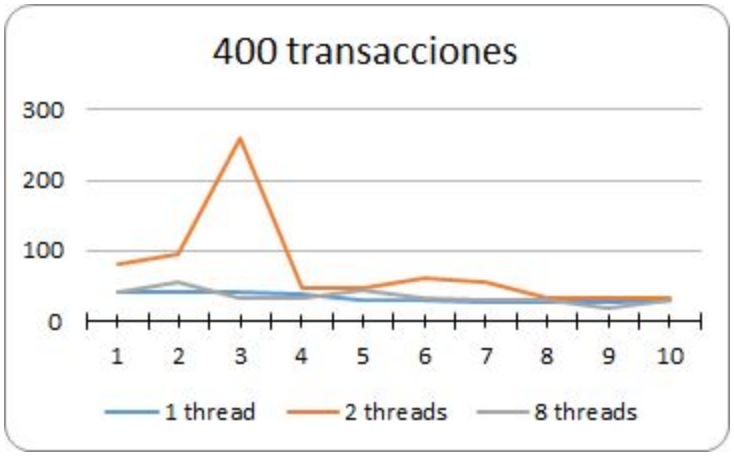
Resultados esperados

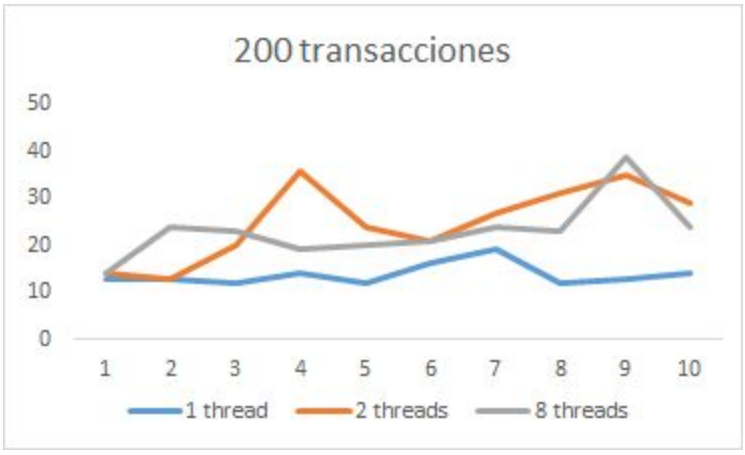
Se espera que la aplicación que no implemente funciones de seguridad tenga tiempos más cortos tanto en la autenticación como en la consulta. Esto se debe a que la autenticación en este caso involucra solo el intercambio de cadenas simples de caracteres y también la consulta, mientras que en aquella que sí implementa funciones de seguridad, se manda un certificado que requiere mayor complejidad para su verificación, así como una consulta cifrada, que debe ser verificada en cuanto a la integridad del mensaje. Por esta misma razón es que se espera un menor número de transacciones perdidas y menor uso de la CPU, pues al usar menos tiempo, puede procesar un mayor número de peticiones simultáneamente sin exceder el límite de tiempo designado por el servidor.

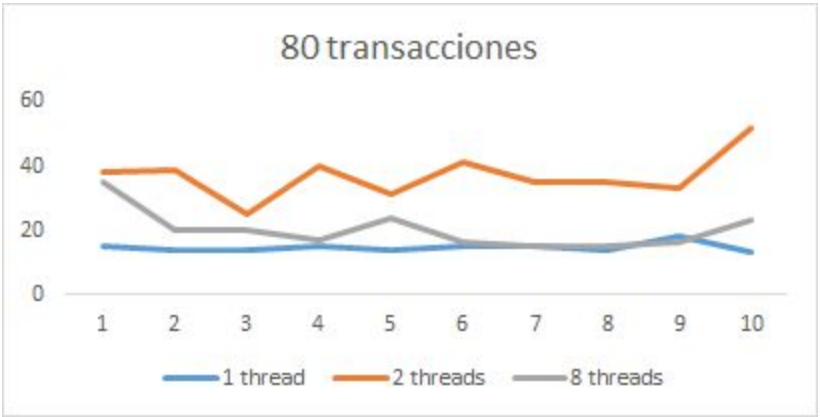
Gráficas

Número de thread vs. tiempo de autenticación

Eje x: # de prueba
Eje y: tiempo de autenticación en milisegundos

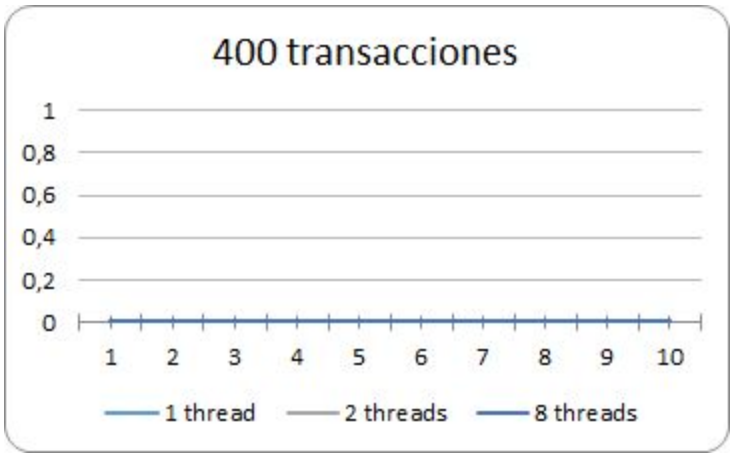


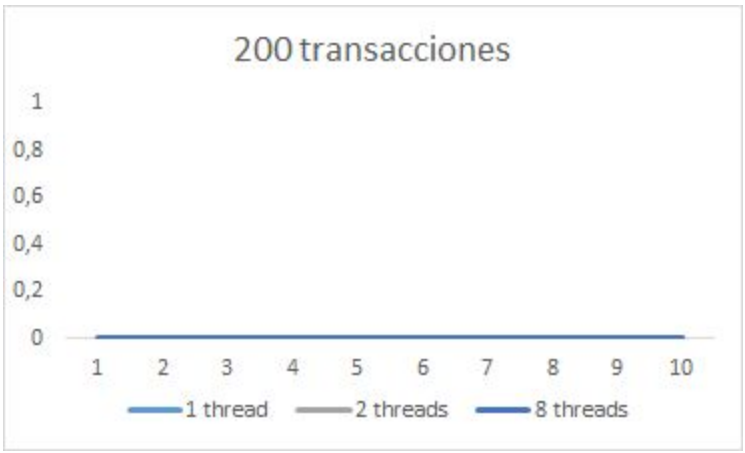


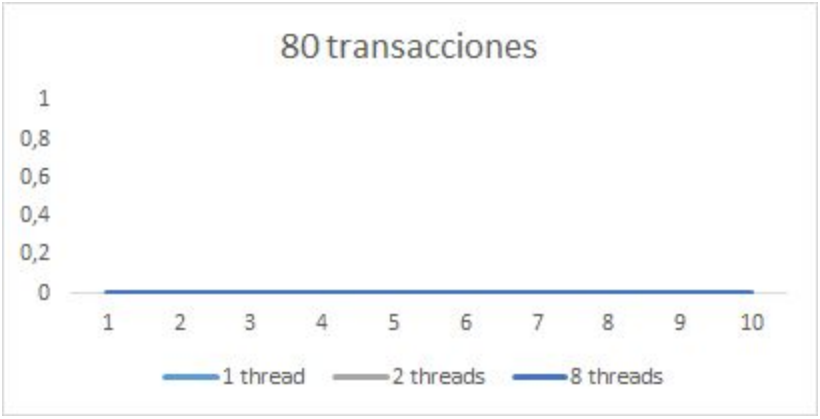


Número de threads vs. número de transacciones perdidas

Eje x: # de prueba
Eje y: #de transacciones perdidas

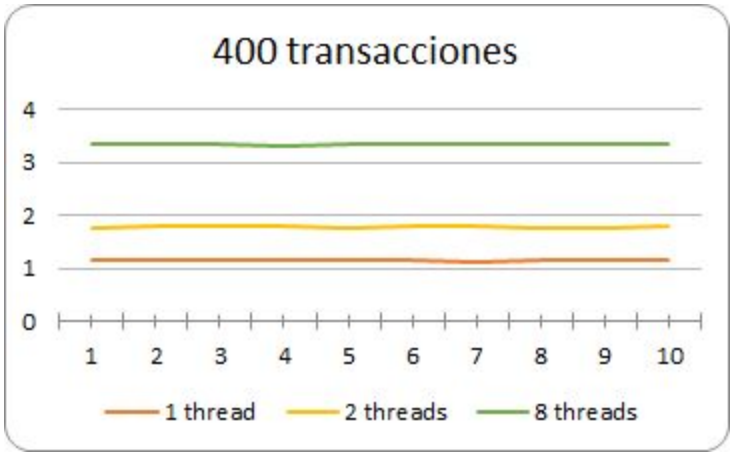


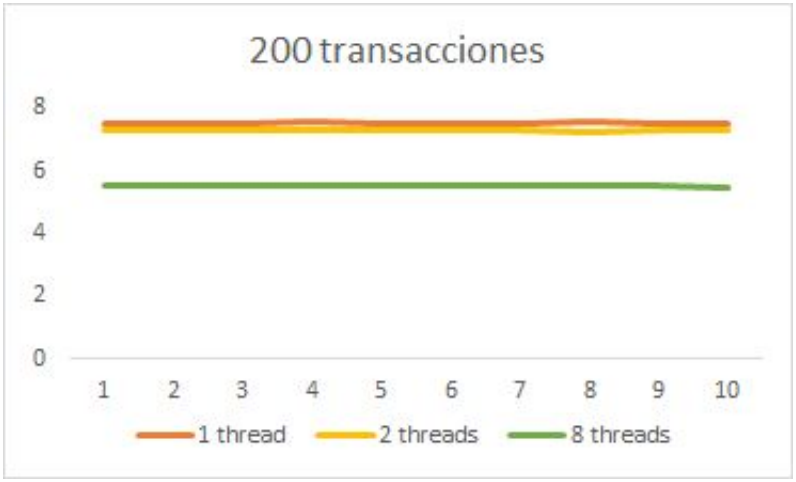


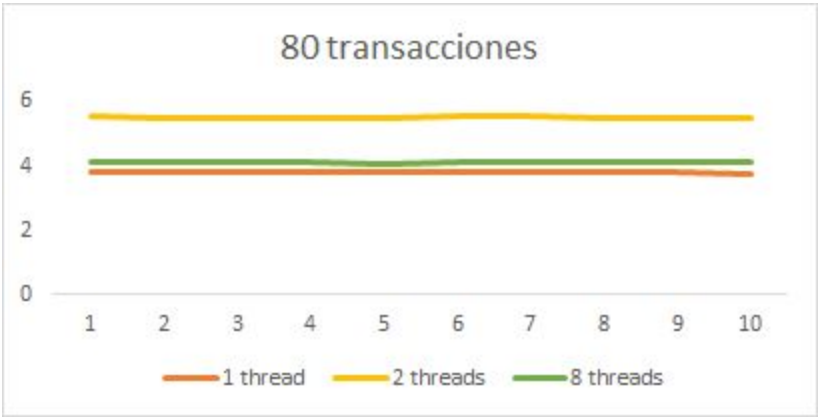


Número de threads vs. porcentaje de uso de la CPU

Eje x: # de prueba
Eje y: % de uso de la CPU

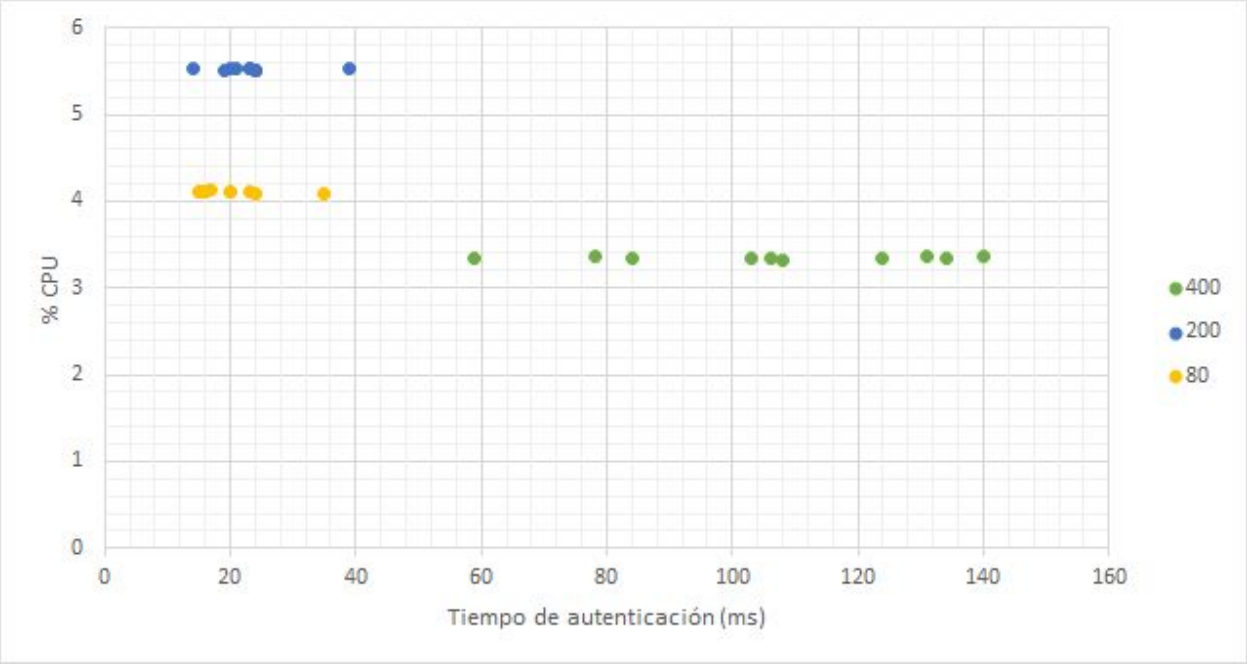

400 transacciones
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10
1 thread
2 threads
8 threads


200 transacciones
8
6
4
2
0
1 2 3 4 5 6 7 8 9 10
1 thread
2 threads
8 threads


80 transacciones
6
4
2
0
1 2 3 4 5 6 7 8 9 10
1 thread
2 threads
8 threads

Tiempo de autenticación vs. porcentaje de uso de la CPU con 2 threads

Tiempo de autenticación vs. porcentaje de uso de la CPU con 8 threads



Conclusiones

Cuando se usa el protocolo sin seguridad todas las variables medidas presentan valores evidentemente menores. Al tener que intercambiar certificados como texto plano y acceder al

resto de las funcionalidades del mismo modo, los tiempos se reducen y el procesador no tiene que ejercer tanto trabajo como en el caso con seguridad.
La simplicidad de este modelo del sistema aparentemente hace que los resultados sean más volátiles. Al comparar número de threads vs. tiempo de autenticación se obtiene un resultado favorable para la configuración con un thread, mientras que la que posee dos threads presenta picos donde toma tiempo de más para realizar la autenticación. En esta prueba es de notar que la configuración de 8 threads se comporta muy parecido a la de 1 thread para el caso de 80 transacciones, sugiriendo un manejo rápido en la distribución de la concurrencia.
Mientras que la tasa de transacciones perdidas se mantiene igual al caso con seguridad por las mismas razones, el porcentaje de uso de la CPU sí presenta un cambio considerable. Resulta ser que el número de transacciones, hace que este porcentaje cambie más dependiendo del número de threads. Para 400 transacciones la configuración de 8 threads es la que mayor porcentaje de cpu consume, y al disminuir el número de transacciones la de dos threads ocupa este lugar (el que más cpu consume). Sin embargo, es de considerar que en todos los casos el porcentaje es muy bajo (nunca supera el 8%), lo que puede indicar que haya otras variables que no son tenidas en cuenta que influyen en el resultado, que termina siendo muy cercano para las configuraciones presentadas. Con respecto al tiempo de autenticación vs porcentaje de cpu se aprecia que hay mas dispersión en los tiempos de autenticación pero el porcentaje de cpu se mantiene constante para todas las iteraciones y al igual que en el esquema con seguridad los pool de threads con 2 y 8 presentaron tiempos considerablemente menores al pool de 1 thread.