Informe y análisis del caso III

Infraestructura Computacional
Caso III
Christian Chavarro Espejo 201613724
Juan Sanmiguel Mateus 201617603

Descripción detallada de los cambios realizados

Tiempo de creación llave:

Este tiempo se mide desde que el cliente envía la solicitud al servidor, hasta que el cliente recibe la llave. El tiempo resultante, se escribe en un log en el cual se registran todos los tiempos de obtención de la llave asociados a la prueba.

```
        continue;
    }
    else if(pasoActual ==4)
    {
        System.out.println("Cliente:("+pasoActual+")" + fromUser);
        escritor.println(fromUser);
        medicion1a=System.currentTimeMillis();

    }

    //Cifra las coordenadas con la llave simetrica.
```

Inicia el tiempo de conteo para la obtención de la llave.

```
    }
    else if(pasoActual == 5)
    {
        //Se recibe la llave por parte del servidor.
        medicion1b=System.currentTimeMillis();
        obtencionL=medicion1b-medicion1a;
        String[] partes = fromServer.split(":");
        if(partes.length > 1)
        {
            byte[] decodedKey = DatatypeConverter.parseHexBinary(partes[1]);
            // rebuild key using SecretKeySpec
            String param = "RSA";
            param = param + ((param.equals("DES")) || (param.equals("AES")) ? "/ECB/PKCS5Pa
            Cipher localCipher = Cipher.getInstance(param);
            localCipher.init(2, keyPair.getPrivate());
            byte [] ls = localCipher.doFinal(decodedKey);
            desKey = new SecretKeySpec(ls, 0, 16, "AES");
            pasoActual++;
        }

    }
```

Finaliza el tiempo de conteo para la obtención de la llave.

Tiempo de respuesta servidor:

Este tiempo se mide desde que el cliente envía ACT1 al servidor, hasta que el cliente recibe la respuesta del servidor, tras haber enviado ACT2 también. El tiempo resultante, se escribe en un log en el cual se registran todos los tiempos de obtención de la llave asociados a la prueba.

```
//Cifra las coordenadas con la clave simetrica.
//Envia al servidor las coordenadas cifradas, las envia bajo el protoc
else if(pasoActual==6)
{
    medicion2a=System.currentTimeMillis();
    byte[] mensaje = simetrico.cifrar(coordenadasPrueba, desKey);
    String msg= ACT1 + DatatypeConverter.printHexBinary(mensaje);

    escritor.println(msg);
    System.out.println("Cliente:("+pasoActual+")" + msg);
    continue;
}

//Cifra el resultado de la función de hash de las coordenadas.
```

Se inicia el conteo del tiempo de respuesta del servidor.

```
//Notifica el fin, cierra el ciclo de ejecución.
else if(pasoActual ==8)
{
    medicion2b=System.currentTimeMillis();
    tiempo=medicion2b-medicion2a;
    escribirLog();
    System.out.println("Fin");
    escritor.println(fromUser);
    ejecutar =false;
}
```

Fin del conteo del tiempo de respuesta del servidor.

Método de escritura en archivos

Este método, escribe los tiempos de obtención de la llave y de respuesta del servidor en un archivo de texto.

Los logs se pueden encontrar en la carpeta docs de cada proyecto.

```
public  void escribirLog() throws IOException
{

    File f1 = new File("./docs/pruebas80/8hilos/prueba10/llave.txt");
    File f2 = new File("./docs/pruebas80/8hilos/prueba10/respuestaServer.txt");


    FileWriter fw1 = new FileWriter(f1.getAbsoluteFile(),true);
    BufferedWriter bw1 = new BufferedWriter(fw1);
    bw1.append(obtencionL+"");
    bw1.append("\r\n");
    bw1.close();

    FileWriter fw2 = new FileWriter(f2.getAbsoluteFile(),true);
    BufferedWriter bw2 = new BufferedWriter(fw2);
    bw2.append(tiempo+"");
    bw2.append("\r\n");
    bw2.close();
}
```

Cantidad de solicitudes perdidas: Este número, se calcula restando del total de transacciones enviadas, el total de transacciones que presentan tiempos de respuesta.

Porcentaje de uso de CPU por parte del servidor: Este indicador, se obtuvo mediante una librería de Java. La clase MBeanServerConnection y OperatingSystemMXBean. Proveen estos servicios.

```
public static final boolean SHOW_OUT = true;
private MBeanServerConnection mbsc = ManagementFactory.getPlatformMBeanServer();
private OperatingSystemMXBean osMBean =null;
// algoritmos
```

Creación de los atributos que facilitaran la medida del porcentaje de uso de la CPU.

```
    }
    por=(int)(osMBean.getProcessCpuLoad()*100);

        try {
            escribirLog();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

       por=0;
}
```

Conteo del porcentaje de uso de la CPU, se encuentra en el método run de la clase worker.

```
    }
    public void escribirLog() throws IOException
    {

        File file1= new File("./docs/pruebas80/8hilos/prueba10/cputime.txt");
        FileWriter fw1= new FileWriter(file1.getAbsoluteFile(),true);
        BufferedWriter bw= new BufferedWriter(fw1);
        String p= por+"";
        bw.append(p);
        bw.append("\r\n");

        bw.close();
    }
```

Método que escribe en un log el porcentaje de uso del procesador del proceso.

Los logs se pueden encontrar en la carpeta docs de cada proyecto.

Identificación de la plataforma. Defina la máquina en la que correrá el servidor e identifique las siguientes características. Tenga en cuenta que el servidor deberá correr en la misma máquina para todos los experimentos (si cambia de máquina los resultados no serán comparables).

Maquina servidor

- Arquitectura 64
- Núcleos 4 (8 subprocesos)
- Velocidad 3,4-3,8 GHz
- RAM 16gb.
- Asignado 12 gb.

Maquina cliente

- Arquitectura 64
- Núcleos 4 (8 subprocesos)
- Velocidad 3,4-3,9GHz
- RAM 8gb
- Asignado 6gb

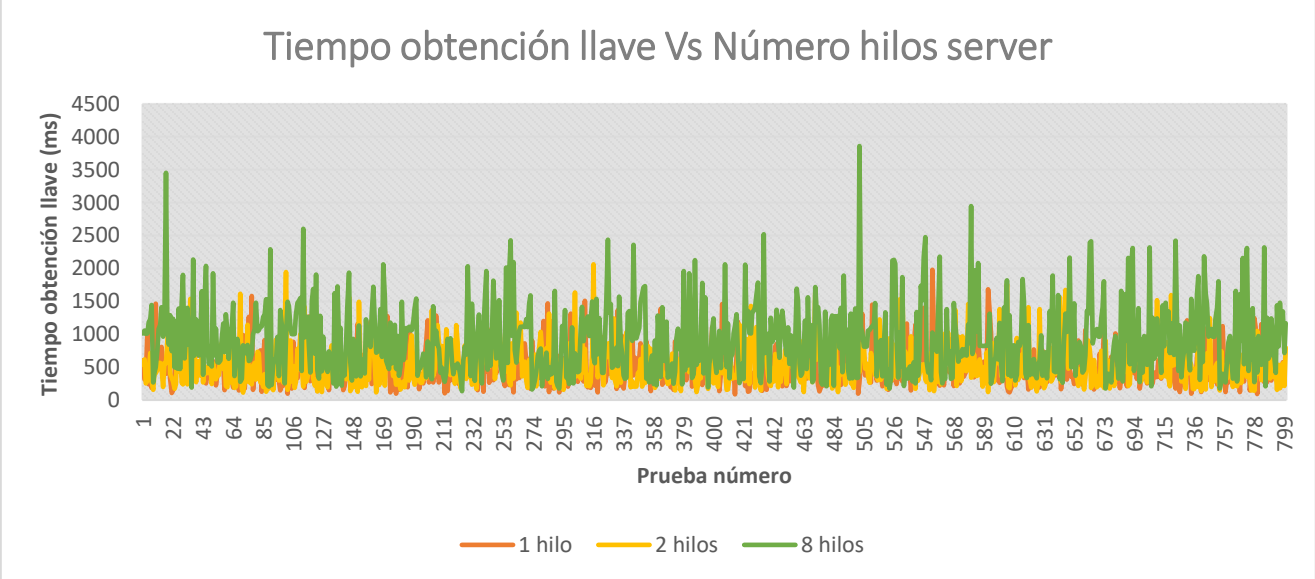
Comportamiento de la aplicación con diferentes estructuras de administración de la concurrencia. Este escenario tiene como objetivo evaluar cambios en el comportamiento del servidor ante diferencias en el manejo de la concurrencia, para esto:

- Revise el código del servidor para que implemente un pool de threads de tamaño definido y asigne threads por conexión.
- Genere escenarios diferentes cambiando el valor de las variables número de threads en el pool y carga. Para la primera variable (número de threads) use los valores 1, 2 y 8. Para la segunda variable (carga) use 400 transacciones iniciadas con retardos de 20 ms, 200 transacciones iniciadas con retardos de 40 ms y 80 transacciones iniciadas con retardos de 100 ms. (*)
- Para cada uno de los 9 escenarios posibles (3 tamaños de pool * 3 tipos de carga) mida los siguientes indicadores: tiempo para creación de la llave simétrica, tiempo para actualización, número de transacciones perdidas y porcentaje de uso de la CPU en el servidor. Repita cada experimento 10 veces.
- Cree las siguientes gráficas:

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)

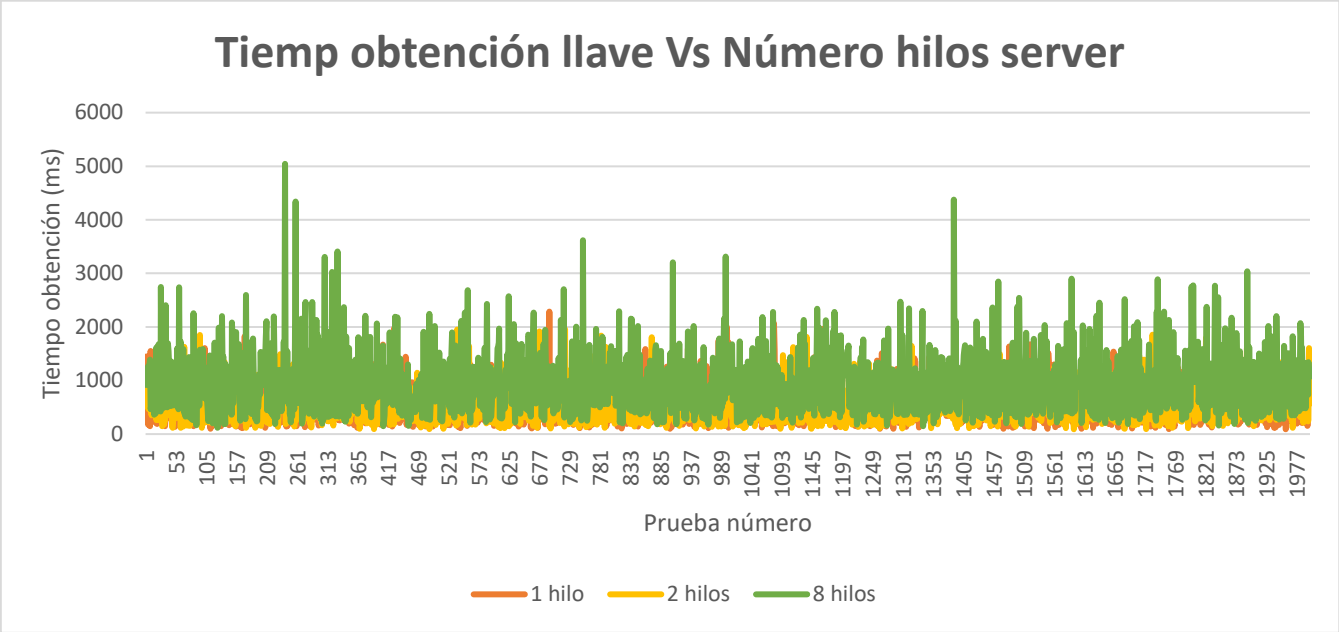
Nota: los datos sobre los cuales se generaron las gráficas, se encuentran en la carpeta gráficas, perteneciente a la misma carpeta de este documento.

Carga 80



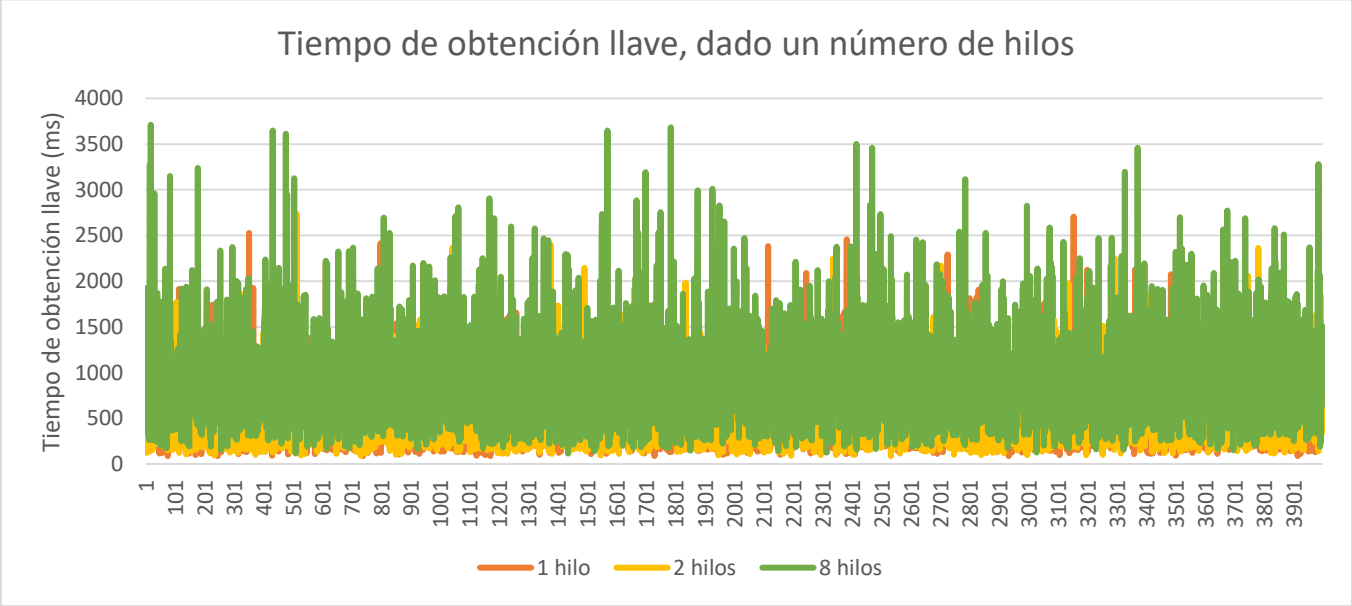
Para una carga baja, como lo son 80 solicitudes, se esperaba que los tiempos de obtención de llave fueran bajos y se hallarán en el rango esperado. En el caso de 8 hilos, se presentan picos a la altura de la solicitud 40-50 en cada prueba.

Carga 200



En una carga de término medio como 200 solicitudes, se esperaba que los tiempos medios se conservarán en el mismo rango que la carga de 80. Esto sucedió, los picos más altos se presentan en el caso de 8 hilos.

Carga 400



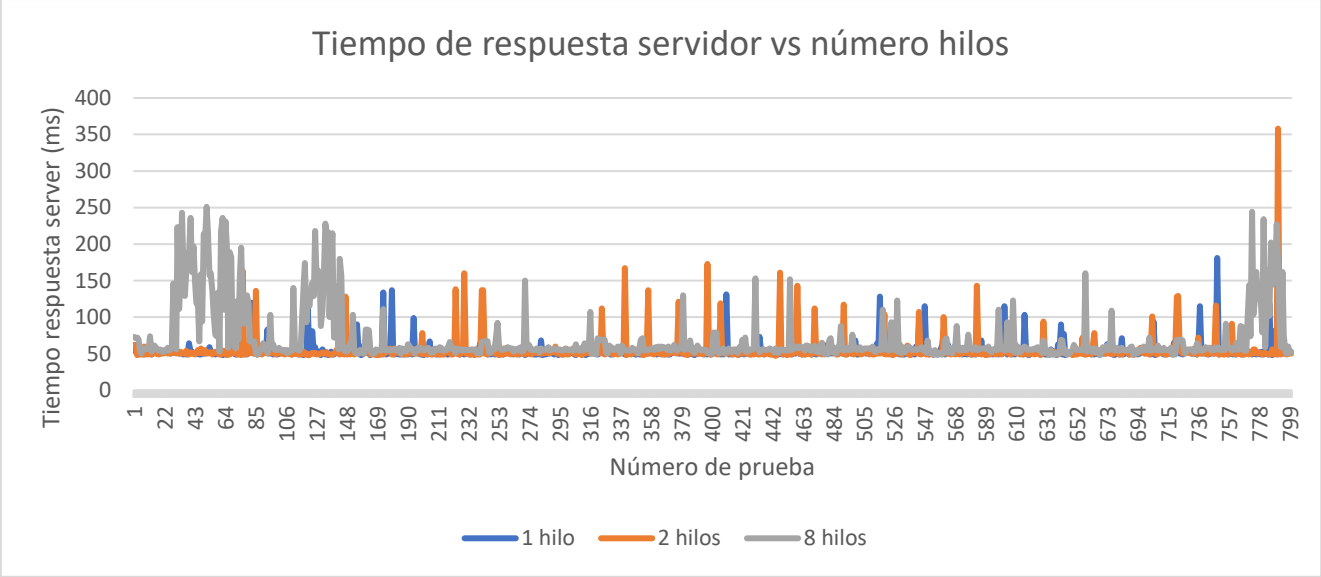
Para un caso de carga alta, como lo son 400 solicitudes, se esperaba que los tiempos de respuesta tuviesen más picos, hecho que sucedió, la media de las respuestas también aumento, pero no drásticamente.

En general, al cambiar la cantidad de carga que se le da al servidor, los tiempos de creación de la llave se ven afectados, mayoritariamente cuando hay más hilos que Cores en la infraestructura. De ahí que en cada gráfica los picos más altos se presenten en la implementación de 8 hilos.

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

Nota: los datos sobre los cuales se generaron las gráficas se encuentran en la carpeta gráficas, perteneciente a la misma carpeta de este documento.

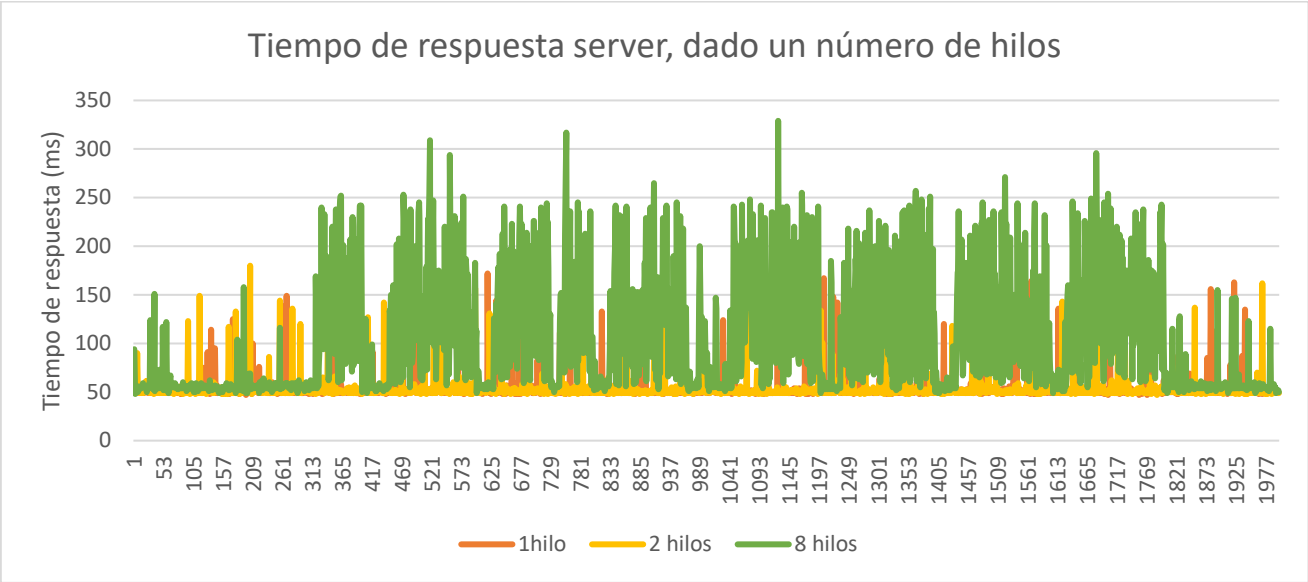
Carga 80



Como se esperaba, el tiempo de respuesta del servidor, es drásticamente menor que el tiempo de obtención de la llave. Esto se puede apreciar con una carga baja y con cargas mayores, como se verá ahora.

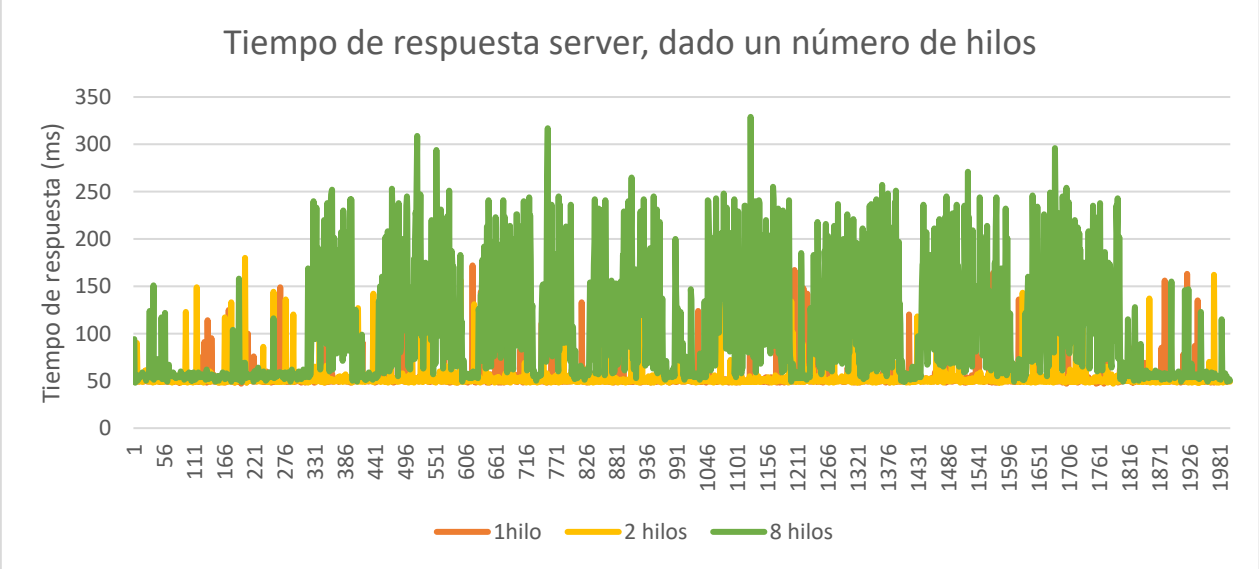
Adicionalmente, cuando se tiene más hilos que cores, los tiempos presentan picos de manera más común.

Carga 200



Al aumentar la carga sobre el servidor, si bien, los tiempos de respuesta se mantienen en el mismo rango, se puede observar como los picos tienen mayor tendencia a ocurrir. Además, los tiempos de respuesta para la implementación de 8 hilos crecieron al compararse con una carga baja, 80 solicitudes por iteración.

Carga 400



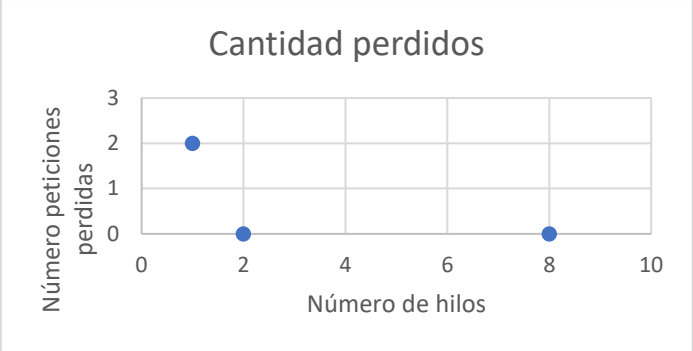
Al aumentar la carga de 80 a 200 solicitudes por iteración, los tiempos de respuesta promedio aumentaron, pero se mantuvieron estables y la tendencia de los picos se hizo mayor. Estas características, se ven reflejadas también en una carga de 400 solicitudes por iteración.

En general, cuando el número de solicitudes al servidor aumenta, el tiempo de respuesta del mismo se verá degradado de acuerdo con la carga que tenga el servidor, la cantidad de cores e hilos tenga.

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

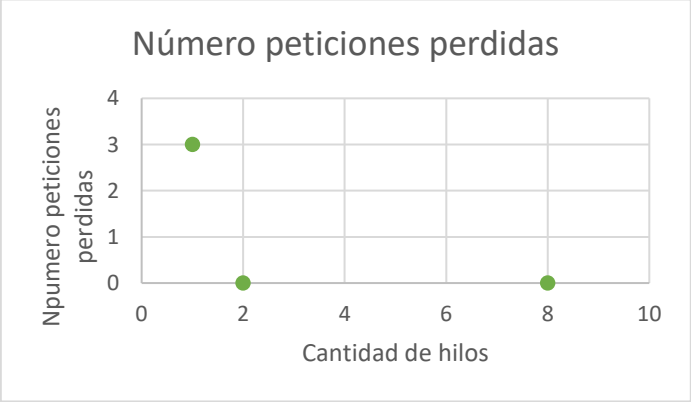
Nota: los datos sobre los cuales se generaron las gráficas se encuentran en la carpeta gráficas, perteneciente a la misma carpeta de este documento.

Carga 80



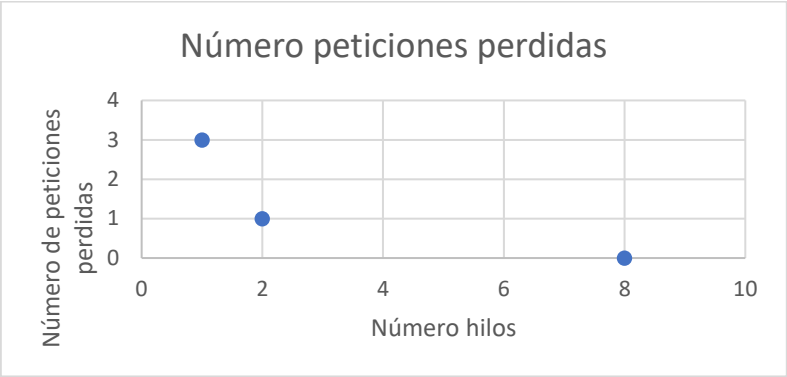
Al haber solo un hilo atendiendo, aumenta la posibilidad de que se pierdan transacciones. Por este hecho, del total de 800 solicitudes, en el caso de un hilo es donde más se presentan errores. Para el caso de dos y ocho hilos, todas las transacciones finalizaron correctamente.

Carga 200



Tal como en el caso de una carga de 80, al tener solo un hilo, la posibilidad de que las transacciones se pierdan aumenta.

Carga 400

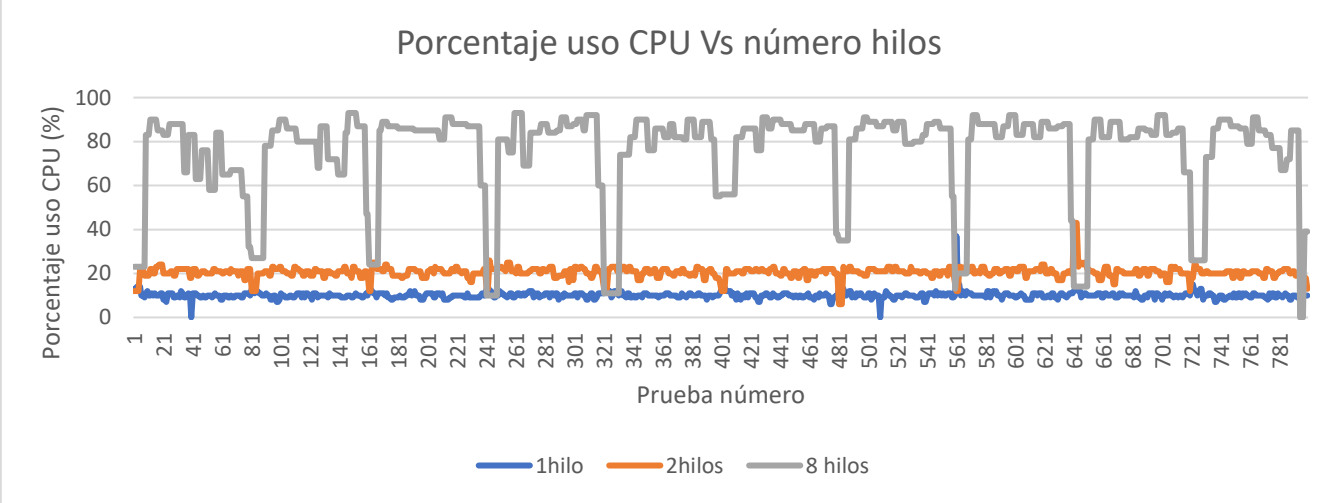


Al tener una carga mayor, las transacciones se empiezan a perder no únicamente en una implementación de un solo hilo, también cuando se tiene más de uno. Así se presencia en la gráfica anterior.

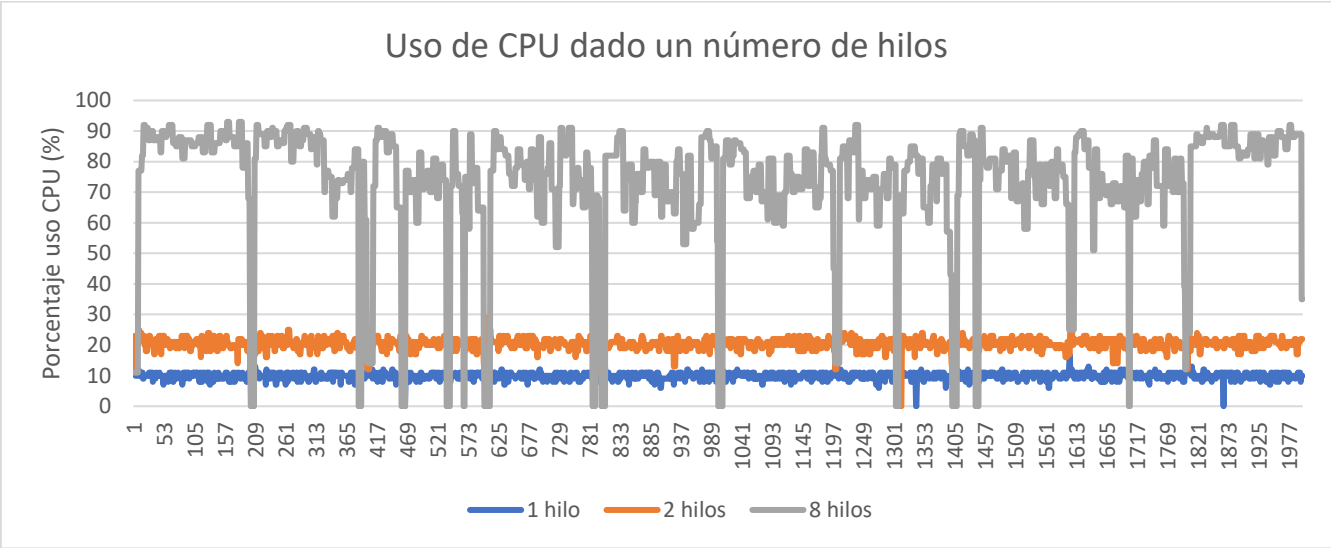
Fije Carga y genere # threads vs. porcentaje de uso de la CPU en el servidor (mismo número de gráficas)

Nota: los datos sobre los cuales se generaron las gráficas se encuentran en la carpeta gráficas, perteneciente a la misma carpeta de este documento.

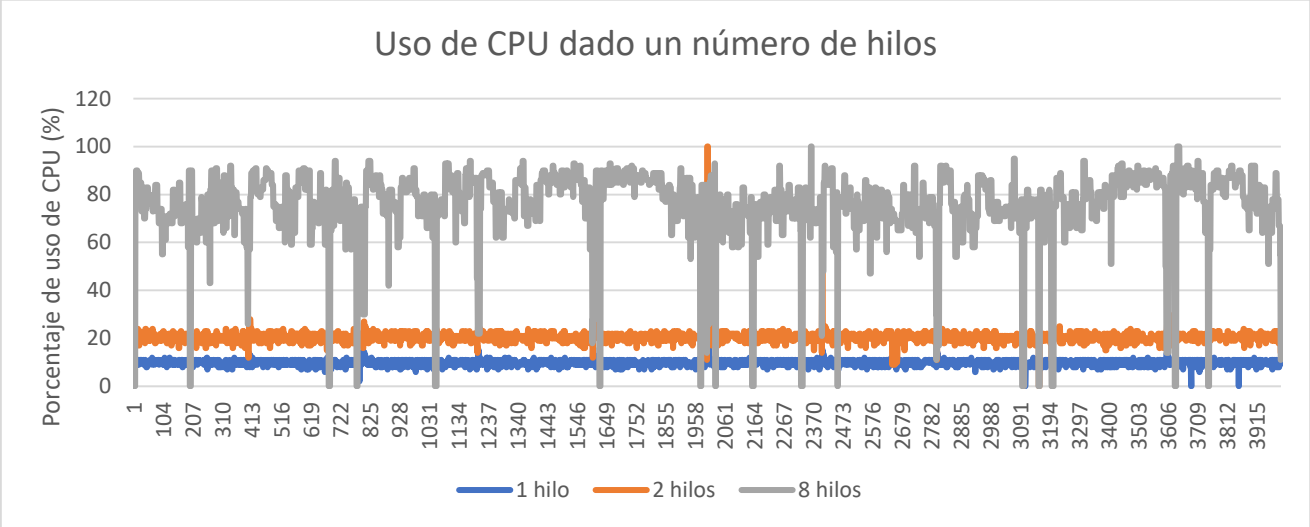
Carga 80



Carga 200



Carga 400



Como se esperaba, para las diferentes cantidades de carga, el porcentaje de uso de la CPU depende de la cantidad de hilos, y presenta una tendencia lineal. En el caso de 8 hilos, el porcentaje de uso es más volátil que en las implementaciones de 1 y 2 hilos, esto, porque requiere mayor uso de recursos de la infraestructura.

- Para cada gráfica analice, compare con otras gráficas y escriba sus conclusiones sobre el comportamiento del sistema ante las diferentes configuraciones.

Comportamiento de la aplicación ante diferentes niveles de seguridad.

- **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.**

Los resultados de estos experimentos se encuentran en la carpeta de este documento, además, las gráficas se encuentran el final de este informe.

- **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?**

Se espera que una aplicación que implementa mecanismos de seguridad tenga mayor consumo de recursos y un tiempo más alto para finalizar las transacciones. Estos tiempos serán mayores, al ser comparados con la misma implementación, misma solución tecnológica, pero que no cuente con mecanismos de seguridad.

- **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.**

Las gráficas que se presentan confirman los resultados esperados enunciados en el anterior literal.

Una implementación sin seguridad para el caso de estudio tendrá menores indicadores en cuanto a tiempos de respuesta, cantidad de transacciones perdidas y porcentaje de uso del procesador porqué:

- La aplicación servidor no debe verificar que el mensaje enviado coincida con la función de hash. No debe realizar los procesos de descifrado para esto.

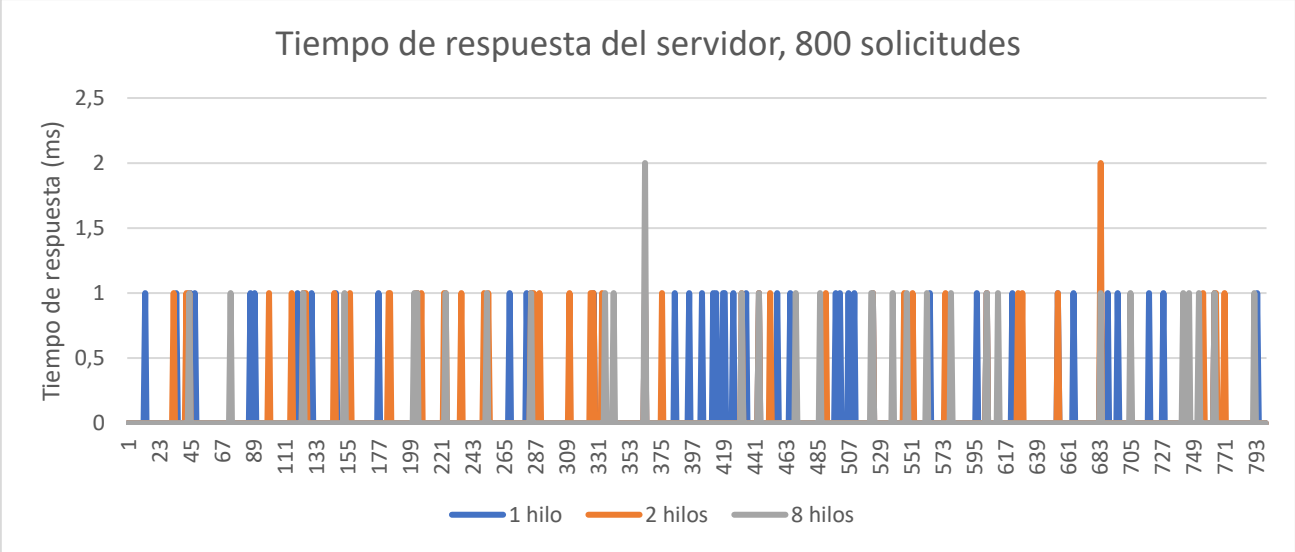
- Al tener peticiones menos complejas, se ejecutan más rápido, disminuyendo la cantidad de transacciones que se pierden. Esto, porqué aumenta la capacidad del servidor para entender solicitudes.

- Como las operaciones son menos complejas, se invierte menos tiempo y recursos de la infraestructura para realizarlas. Por esto, el porcentaje de uso de la CPU es muy bajo en comparación con la implementación con seguridad.

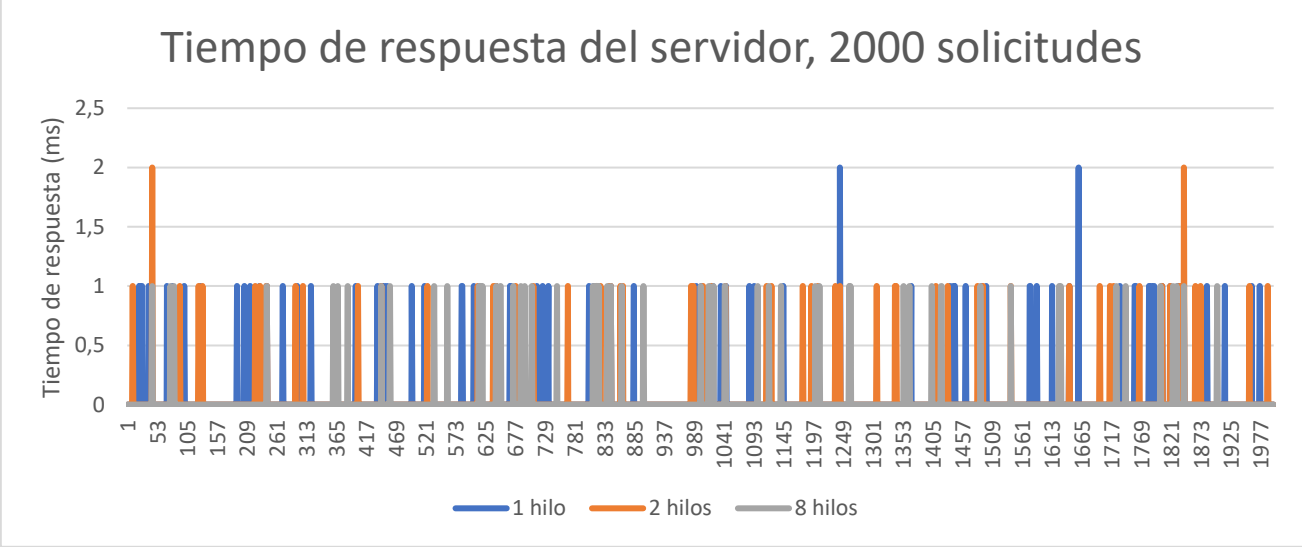
Fije Carga y genere: # threads vs. tiempo de actualización (3 cargas diferentes = 3 gráficas, 3 tamaños de pool 1, 2 y 8: 3 conjuntos de puntos por gráfica)

Nota: los datos sobre los cuales se generaron las gráficas se encuentran en la carpeta gráficas, perteneciente a la misma carpeta de este documento.

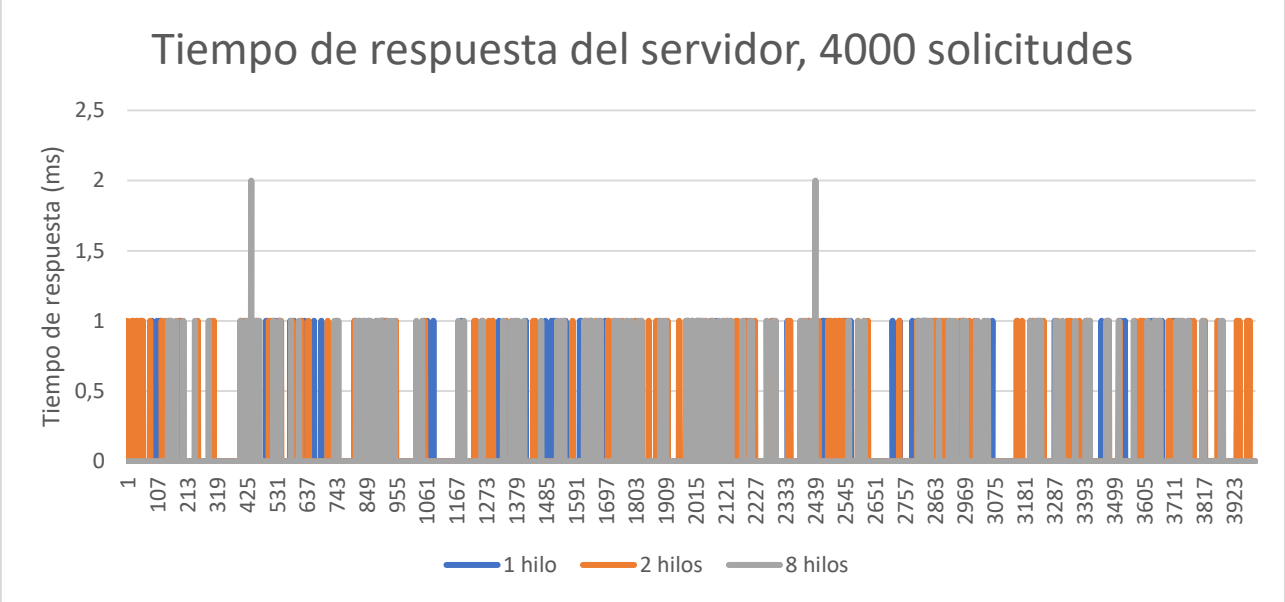
Carga 80



Carga 200



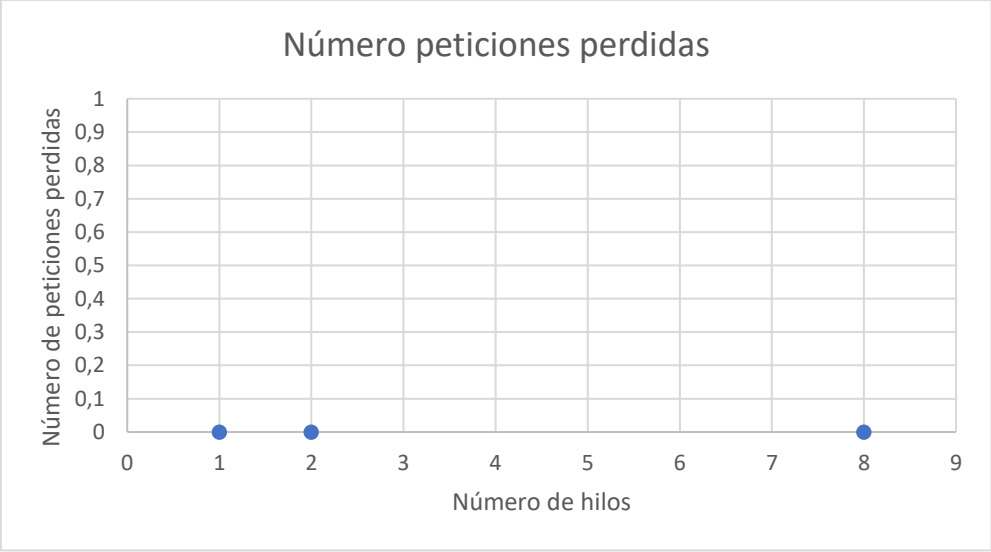
Carga 400



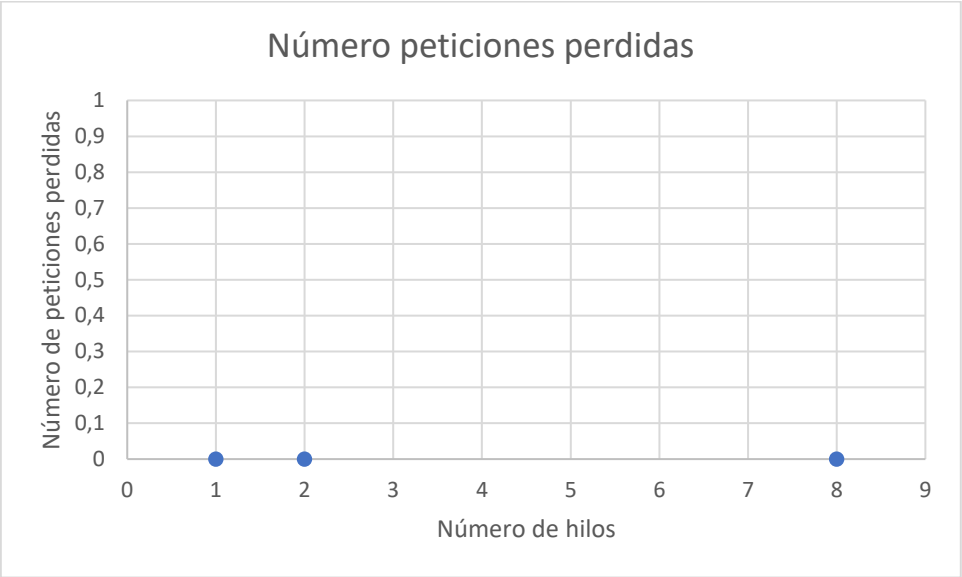
En general, los tiempos de respuesta presentan las mismas características que en una implementación del canal de comunicación con seguridad. Los picos se hacen más frecuentes al aumentar la carga y también aumenta el tiempo promedio si la carga es mayor.

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

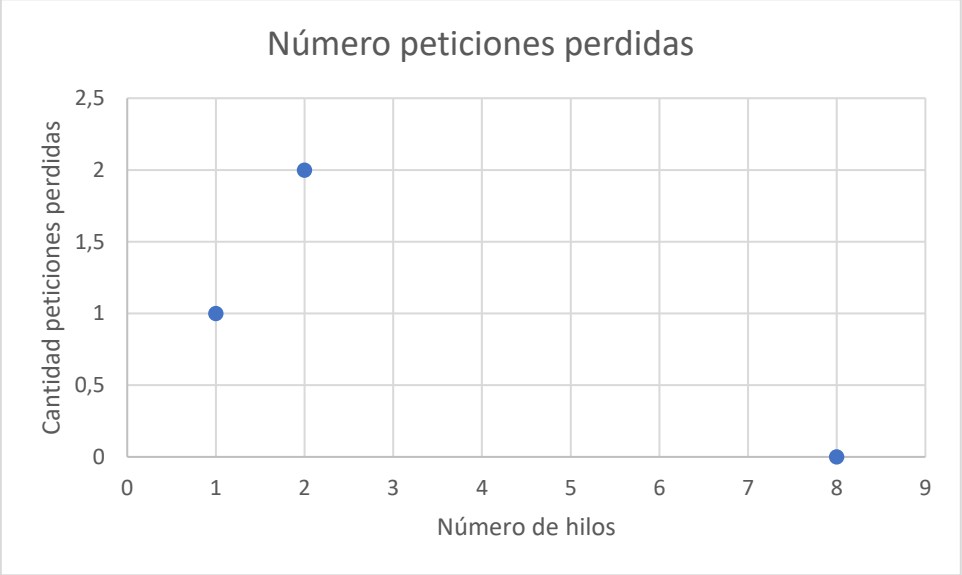
Carga 80



Carga 200



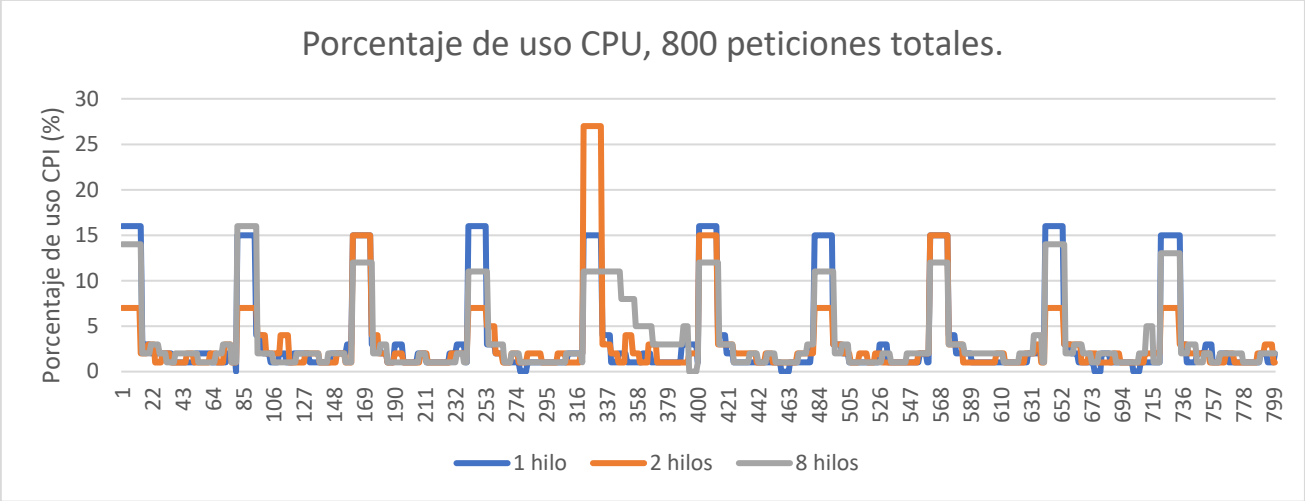
Carga 400



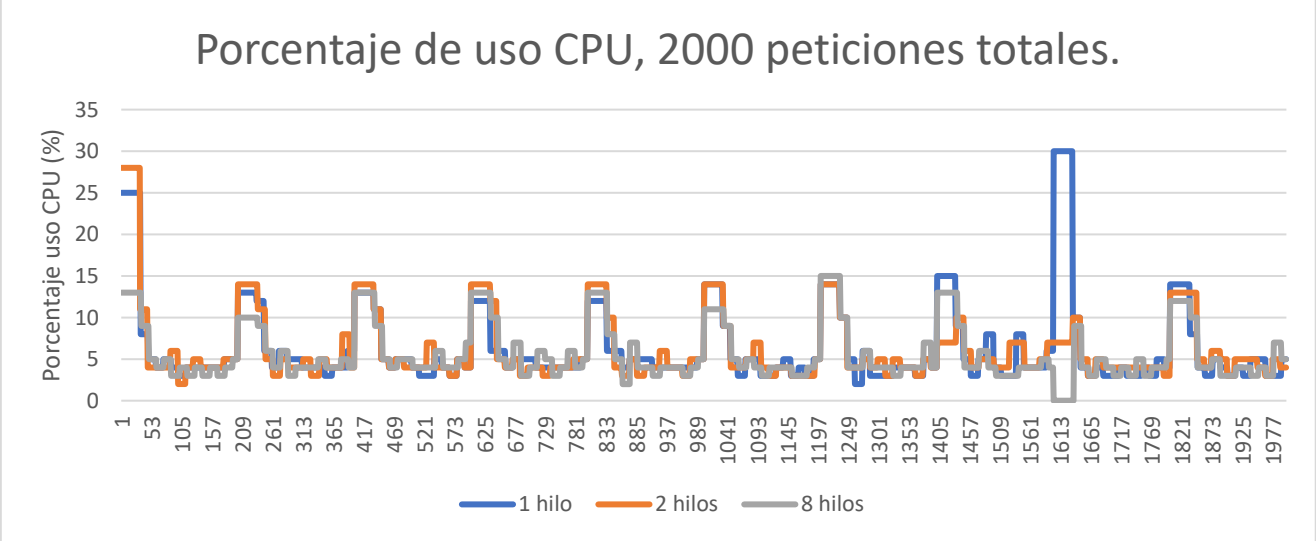
Como la complejidad del proceso a ejecutar por parte del servidor es menor, se empiezan a perder peticiones cuando la carga dada al servidor empieza a ser alta, el primer fallo se presenta con carga de 400 solicitudes.

Fije Carga y genere # threads vs. porcentaje de uso de la CPU en el servidor (mismo número de gráficas)

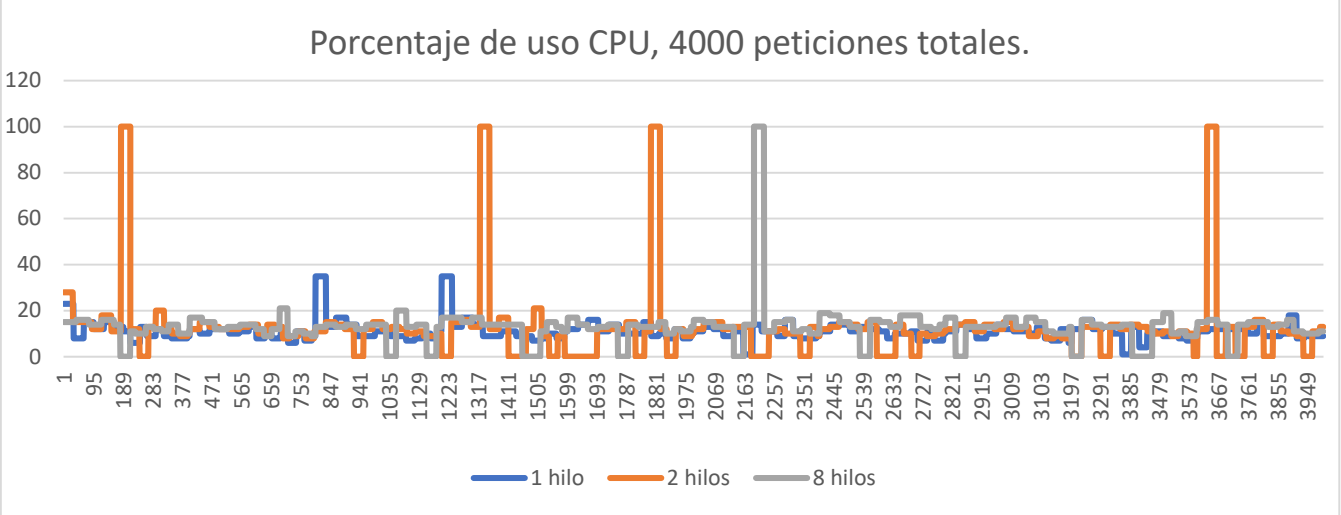
Carga 80



Carga 200



Carga 400



Como el servidor debe procesar transacciones de poca complejidad, la cantidad de procesador a utilizar el baja también, los picos que presentan las gráficas son en momentos en los cuales se iniciaban las peticiones, inicio de las iteraciones sobre las pruebas de carga.