CASO 3

Julián David Mendoza Ruíz - 201730830

María Alejandra Pabón Galindo - 201728807

IMPLEMENTACIÓN DE MONITORES:

Para la implementación de los monitores se decidió almacenar los datos recopilados en archivos CSV, de manera que la manipulación de datos sea más eficiente posteriormente.

Con respecto a la implementación en código, en la clase principal del servidor (C.class) se creó el archivo en el que se escribe los datos recopilados en una misma transacción; este archivo se pasa a la clase del servidor delegado (D.class) como un archivo estático, garantizando que todos los delegados escriban sobre un mismo archivo.

En el archivo CSV cada delegado lo primero escribe el ID correspondiente de la transacción que lo identifica. Para esto la clase del delegado tiene como atributo un ID único (se asigna desde el servidor maestro).

Después de esto, el delegado escribe en el CSV el tiempo total que duró la transacción, desde el momento en que lee y procesa el certificado del cliente hasta leer del socket el OK o ERROR de terminación. Se usa una variable que registra la diferencia de tiempo en milisegundos del sistema desde la media noche del primero de junio de 1970 a ese instante. Se registra dos veces, una al inicio de la transacción y otra al final de la transacción (sea error o éxito); finalmente se restan y se almacena la duración de la transacción. Si llega a haber un error antes de que llegue al punto en que debería empezar a medir el tiempo el tiempo de la transacción será cero. Por otro lado, en caso de haber un error en la transacción después de medir el primer tiempo y antes de llegar al último tiempo, llega a la sección catch de la función y registra el tiempo. Este es el tiempo que se resta para obtener el tiempo que toma la transacción (en milisegundos).

A su vez, el delegado también debe medir el porcentaje de uso del CPU. Se hizo uso de la función presentada en la guía del caso “getSystemCpuLoad()”. Este retorna en porcentaje el uso el CPU en un momento específico como un valor decimal representativo del porcentaje. A lo largo del código mientras se realiza el protocolo correspondiente se hizo un llamado a este método en 4 puntos diferentes, estos valores fueron guardados en una lista para posteriormente compararlos entre sí y escoger el mayor porcentaje registrado a lo largo del código, esto se hizo con el fin de poder escoger cual es el máximo porcentaje de uso que podría llegar a tener una transacción.

Para registrar las transacciones perdidas se hizo uso de una variable auxiliar que inicializaba en 0. Si terminaba la transacción correctamente esta variable seguiría registrando el valor de 0 indicando que no se perdió la transacción; por el contrario, si hubo un problema en esta y se terminó perdiendo dentro de la sección catch del delegado se cambia el valor de la variable a 1, indicando la pérdida de la transacción y registrándolo. Se eligió esto en vez de un valor booleano ya que al analizar los datos dentro del archivo CSV sería más fácil encontrar cuantas transacciones fueron perdidas en total.

IDENTIFICACIÓN DE LA PLATAFORMA:

- Arquitectura: arquitectura de 64 bits.

- Número de núcleos: 2 núcleos.
- Velocidad del procesador: 2.6 GHz
- Tamaño de la memoria RAM: 4GB.
- Espacio de la memoria asignado a JVM: 1073.74 MB

COMPORTAMIENTO DE LA APLICACIÓN CON DIFERENTES ESTRUCTURAS DE ADMINISTRACIÓN DE CONCURRENCIA:

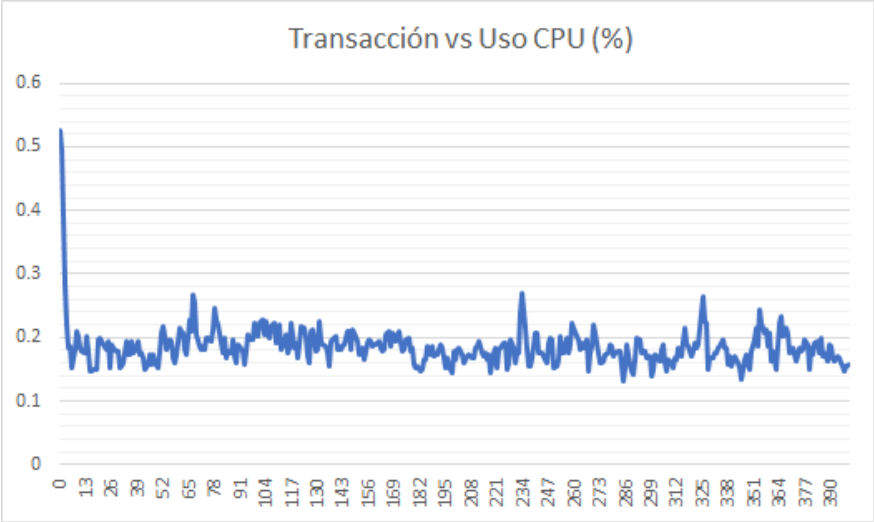
Como se indica en la guía de trabajo se tomaron todos los datos necesarios para los 6 diferentes escenarios presentados. Fueron compilados los datos en sus respectivas carpetas y con estos se hicieron los análisis respectivos.

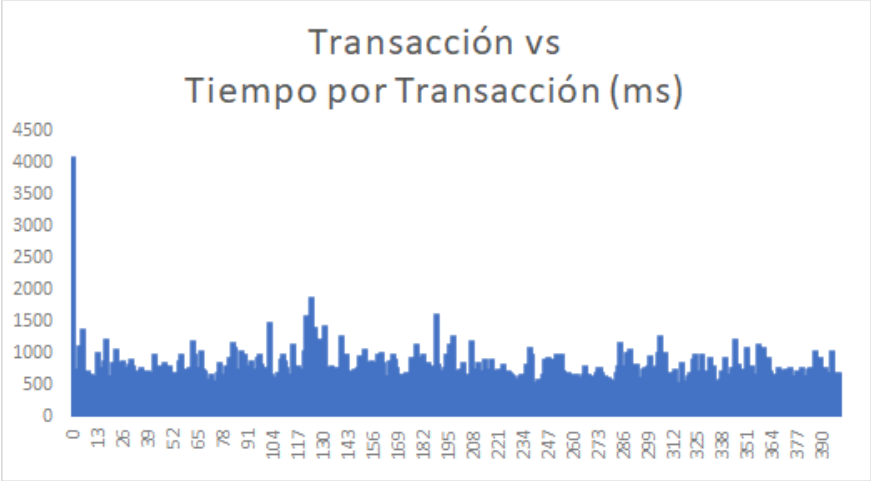
Calidad de los datos: En la toma de datos para evitar ruido de alguna otra aplicación al correr el servidor, se tomó la precaución de correr este en una máquina diferente a la del cliente.

A continuación, se presenta un resumen de cada una de las tomas de datos por medio de sus medidas de tendencia central, medidas de dispersión y gráficas de los valores importantes (Tiempo por transacción, %uso de CPU y número de transacciones fallidas)

1 Thread, 400 transacciones con 20 ms de retraso entre cada una

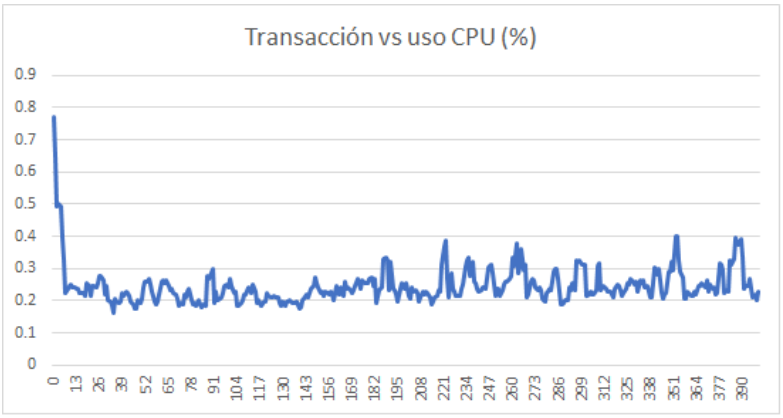
Medidas de tendecia y dispersión	Tiempo respuesta (ms)	Uso CPU (%)	Transacciones perdidas	Transacciones exitosas
Media	737.45	18.52%	0.00	1.00
Mediana	685.50	18.20%	0.00	1.00
Moda	653.00	#N/A	0.00	1.00
Varianza	66566.67	0.10%	0.00	0.00
Desviación estandar	258.01	3.22%	0.00	0.00

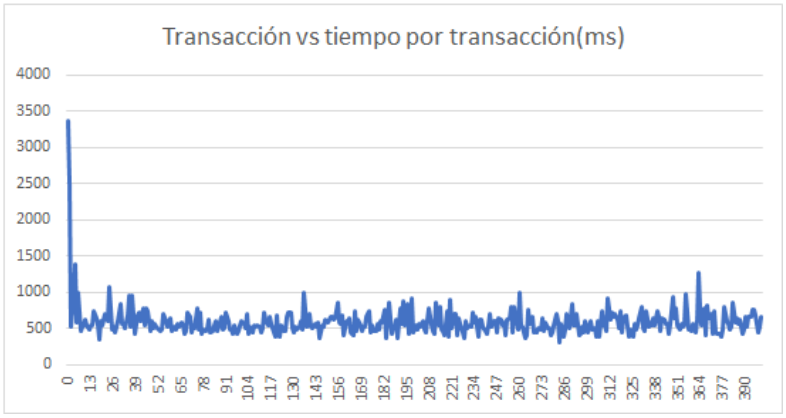




2 Thread, 400 transacciones con 20 ms de retraso entre cada una

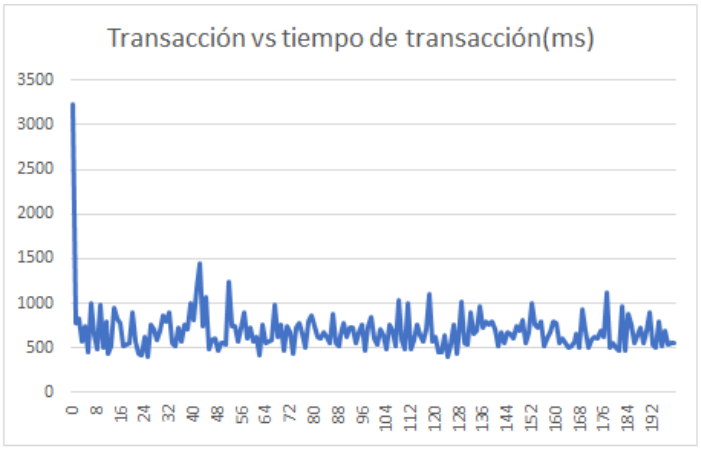
Medidas de tendecia y dispersión	Tiempo respuesta (ms)	Uso CPU (%)	Transacciones perdidas	Transacciones exitosas
Media	590.44	24.63%	0.00	1.00
Mediana	550.00	23.30%	0.00	1.00
Moda	550.10	39.91%	0.00	1.00
Varianza	48068.58	0.34%	0.00	0.00
Desviación	219.25	5.84%	0.00	0.00

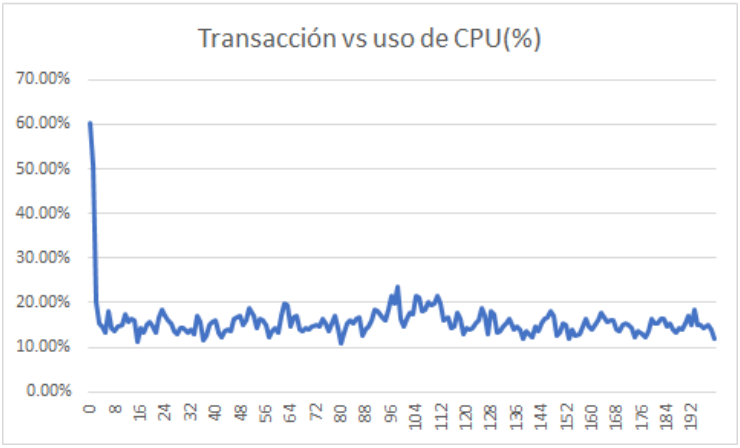




1 Thread, 200 transacciones con 40 ms de retraso entre cada una

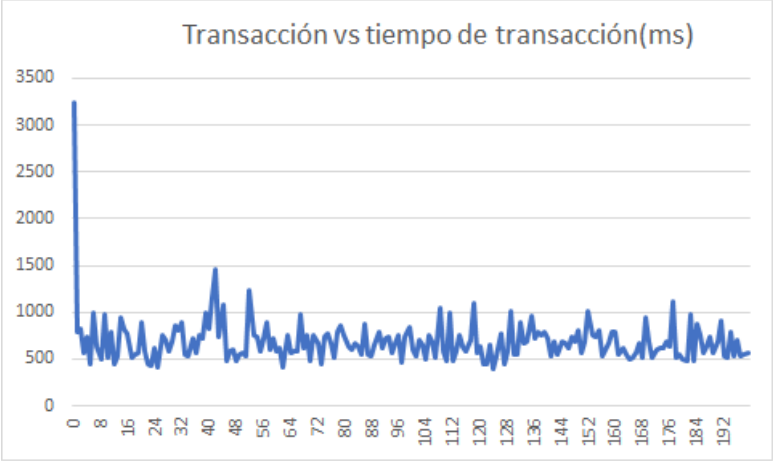
Medidas de tendecia y dispersión	Tiempo respuesta (ms)	Uso CPU (%)	Transacciones perdidas	Transacciones exitosas
Media	687.62	15.82%	0.00	1.00
Mediana	647.50	15.14%	0.00	1.00
Moda	549.50	#N/A	0.00	1.00
Varianza	61818.98	0.21%	0.00	0.00
Desviación	248.63	4.56%	0.00	0.00

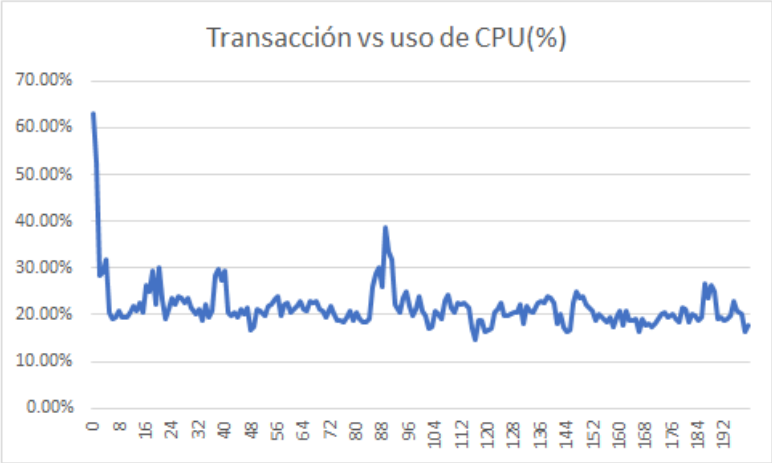




2 Thread, 200 transacciones con 40 ms de retraso entre cada una

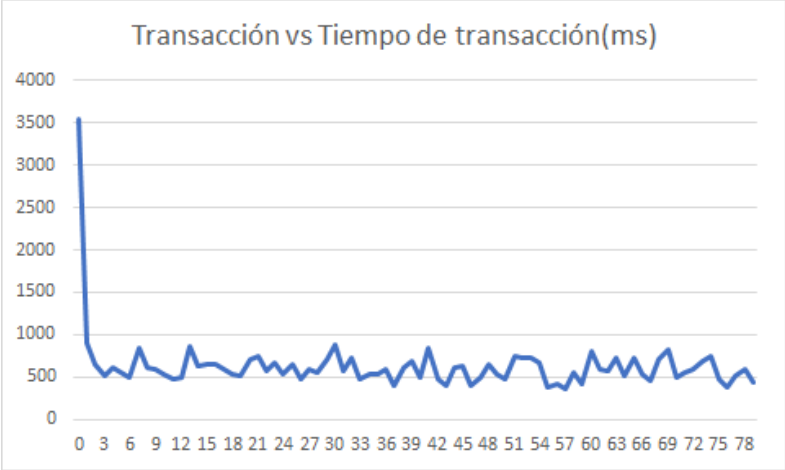
Medidas de tendecia y dispersión	Tiempo respuesta (ms)	Uso CPU (%)	Transacciones perdidas	Transacciones exitosas
Media	666.45	21.72%	0.00	1.00
Mediana	601.25	20.80%	0.00	1.00
Moda	500.10	#N/A	0.00	1.00
Varianza	69868.69	0.25%	0.00	0.00
Desviación	264.33	4.99%	0.00	0.00

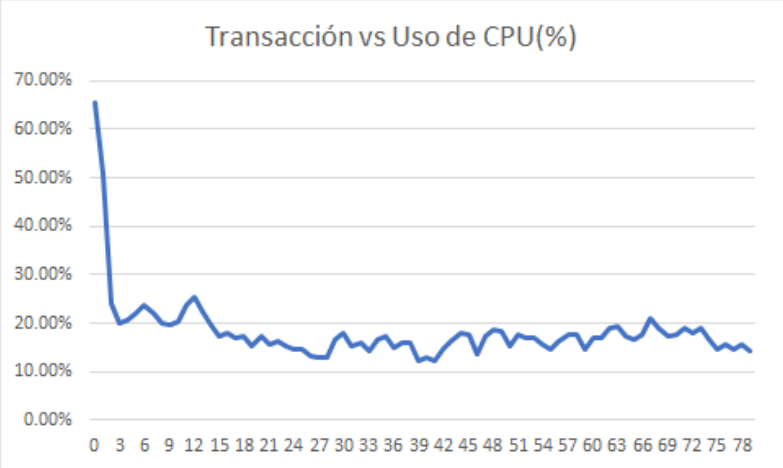




1 Thread, 80 transacciones con 100 ms de retraso entre cada una

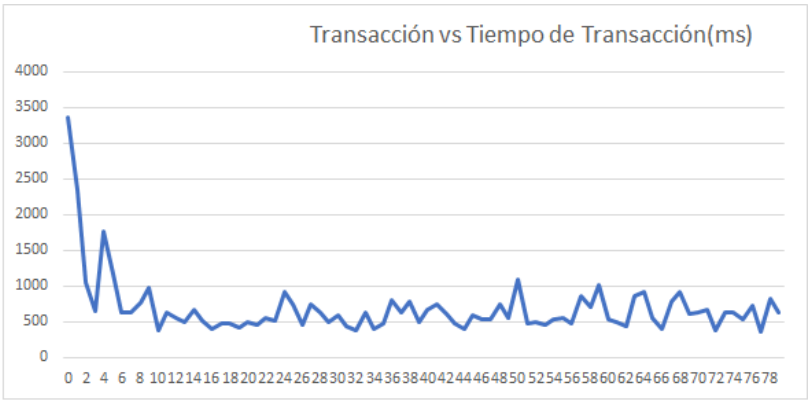
Medidas de tendecia y dispersión	Tiempo respuesta (ms)	Uso CPU (%)	Transacciones perdidas	Transacciones exitosas
Media	631.39	18.26%	0.00	1.00
Mediana	584.95	17.16%	0.00	1.00
Moda	#N/A	#N/A	0.00	1.00
Varianza	123793.42	0.50%	0.00	0.00
Desviación	351.84	7.09%	0.00	0.00

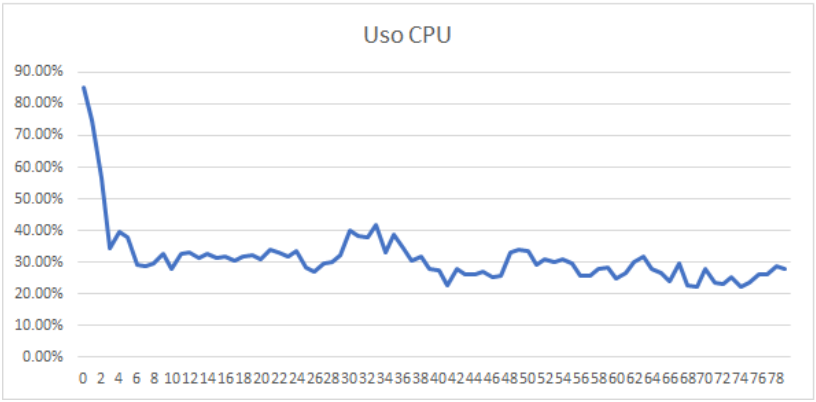




2 Thread, 80 transacciones con 100 ms de retraso entre cada una

Medidas de tendecia y dispersión	Tiempo respuesta (ms)	Uso CPU (%)	Transacciones perdidas	Transacciones exitosas
Media	685.17	31.47%	0.00	1.00
Mediana	599.60	29.88%	0.00	1.00
Moda	#N/A	#N/A	0.00	1.00
Varianza	176999.70	0.89%	0.00	0.00
Desviación estandar	420.71	9.42%	0.00	0.00



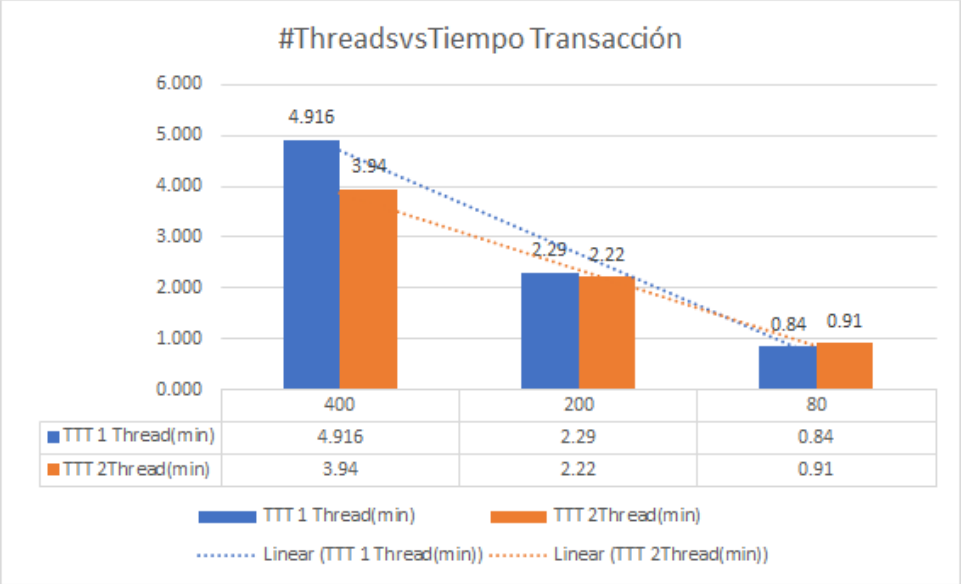


Con los resultados recolectados se observa un comportamiento similar: Para el porcentaje de uso del CPU y tiempo de transacción las primeras transacciones (2-4) tienen el pico más elevado de todos los datos, luego estos valores bajan y permanecen oscilando en un mismo rango, sin embargo, pese a tener el mismo comportamiento almacenan datos en rangos distintos. A sí mismo, no hay distorsión en los datos como causa de la perdida de estos. Se puede observar que para cualquier caso nunca hay pérdida de transacciones, generando así una efectividad del 100% de la aplicación. Cuando se analizan los datos como un solo contexto se puede establecer que hay indicios de que son confiables, ya que sin importar la carga su comportamiento por transacción es similar aún si los valores en si son mayores o menores.

Análisis de gráficas:

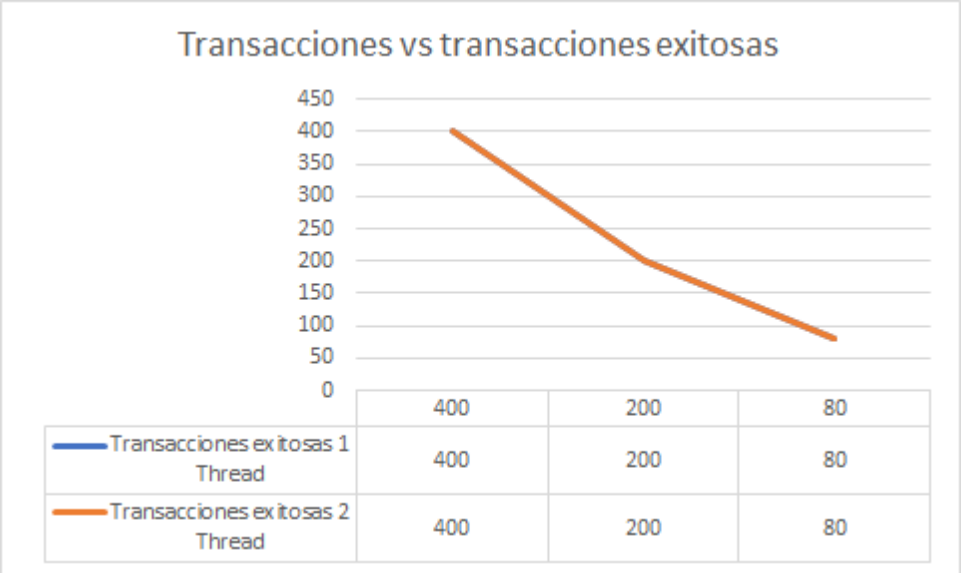
a. # threads vs. tiempo de transacción:

Threads versus Tiempo Total de Transacción		
Transacciones	TTT 1 Thread(min)	TTT 2Thread(min)
400	4.916	3.94
200	2.29	2.22
80	0.84	0.91



Para una carga mayor, 400 transacciones, la diferencia en tiempo de transacción que se toma para el caso de un thread contra otro es considerable (de casi un minuto de diferencia) haciendo para este caso más eficiente el uso de 2 threads en el pool. Para una carga menor, 200 transacciones, la diferencia de tiempo de transacción entre el uso de un thread o dos para el pool de threads no es tan significativa como la carga anterior. Finalmente, para una carga mucho menor, 80 transacciones, termina siendo mejor el tiempo de un pool con un solo thread a con 2 threads. En general, entre más grande sea la carga será mejor usar una mayor cantidad de threads (de igual forma la cantidad de threads tiene un límite del cual una vez es sobrepasado podría tener el efecto contrario por el uso de recursos). Sin embargo, cuando la cantidad de transacciones sea demasiado baja (para esta aplicación que no supere las 200 transacciones) será más eficiente, en tiempo, el uso de un solo thread en el pool ya que los datos parecen indicar que hay un conflicto por el cumplimiento de las tareas entre los threads.

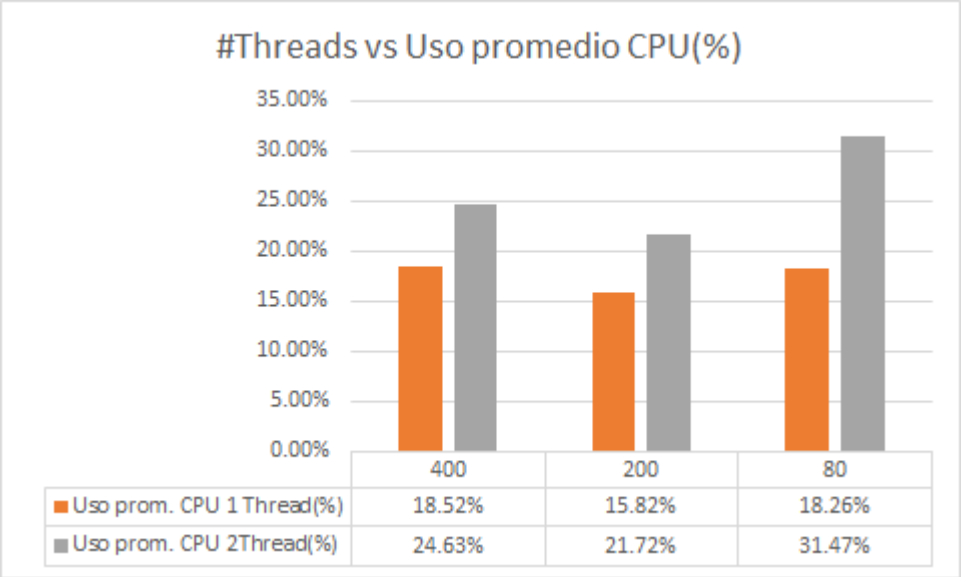
b. # threads vs. # de transacciones exitosas:



De las gráficas se puede observar cómo se presenta una tasa de éxito del 100%, indicando que no hubo ninguna transacción perdida sin importar la cantidad de transacciones (hasta 400 transacciones

con 1 o 2 threads). Esto simboliza que para los escenarios presentados hay una efectividad del 100% por parte de la aplicación al no perderse ninguna de las transacciones, sin importar la carga generada o el tamaño del pool de threads.

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



	400	200	80
Uso prom. CPU 1 Thread(%)	18.52%	15.82%	18.26%
Uso prom. CPU 2Thread(%)	24.63%	21.72%	31.47%

El **comportamiento** general para cualquier carga que se esté usando en la aplicación siempre que se tenga más de un thread va a existir un mayor porcentaje de consumo de CPU por parte del servidor. Esto se debe a que cuando se tiene más de un thread se busca hacer varias tareas en simultaneo. Sin embargo, se observa que este uso no es proporcional ante cualquier carga, ya que para pocas transacciones (en nuestro caso 80) el uso de CPU es mayor. No es posible saber si esto resultado del diseño de los monitores o un problema asociado al uso de threads para cargas bajas, no obstante, deberíamos ver este mismo comportamiento conforme la carga es más baja.

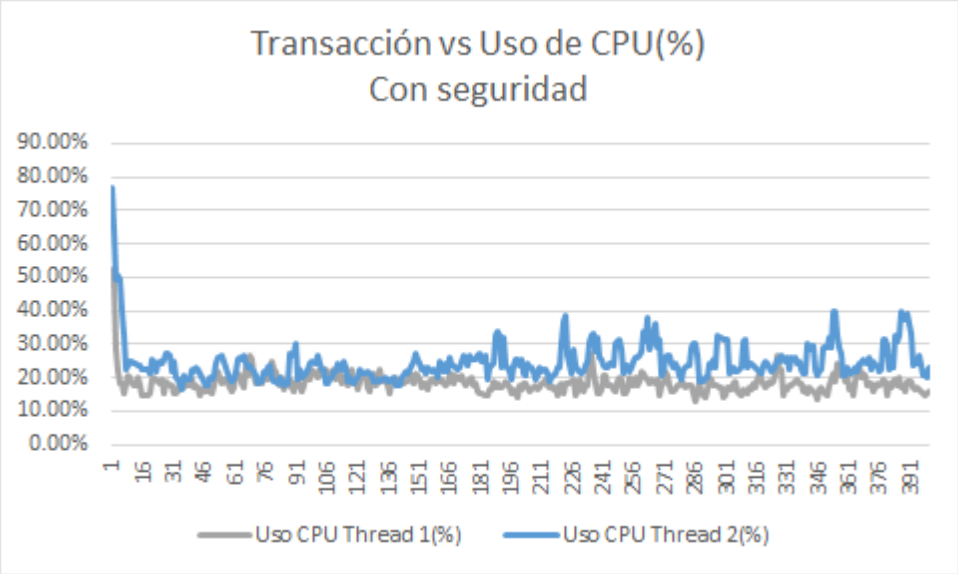
COMPORTAMIENTO DE LA APLICACIÓN CON DIFERENTES NIVELES DE SEGURIDAD:

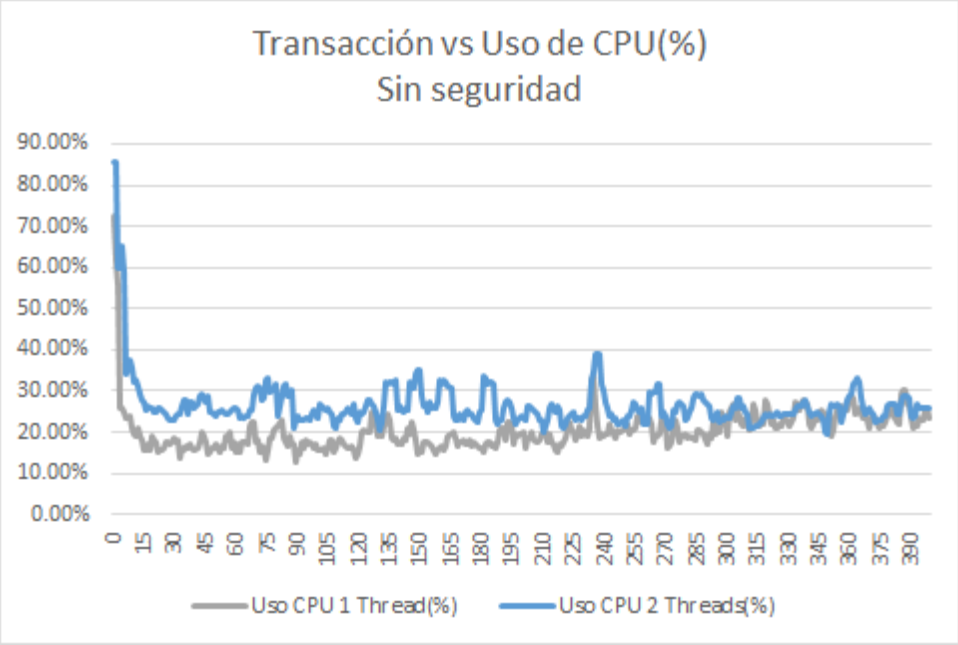
Como se indica en la guía de trabajo se tomaron todos los datos necesarios para los 2 escenarios diferentes presentados ante un protocolo que no implementa seguridad. Se consolidaron los datos en sus respectivas carpetas (conf 7, conf 8) y con estos se hicieron los análisis respectivos.

Calidad de datos: Con los resultados recolectados se observa un comportamiento similar: Para el porcentaje de uso del CPU se tiene el pico más elevado de todos los datos al comienzo, luego estos valores bajan y permanecen oscilando en un mismo rango, sin embargo, pese a tener el mismo comportamiento almacenan datos en rangos distintos. A sí mismo, no hay distorsión en los datos como causa de la perdida de estos. Se puede observar que para cualquier caso nunca hay pérdida de transacciones, generando así una efectividad del 100% de la aplicación. Cuando se analizan los datos como un solo contexto se puede establecer que hay indicios de que son confiables, ya que sin importar la carga su comportamiento por transacción es similar aún si los valores en si son mayores o menores.

Pregunta: ¿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?

R/ Se esperaría que una aplicación que implemente funciones de seguridad haga uso de más recursos y tome más tiempo por transacción a una aplicación que no implemente funciones de seguridad. Esto se debe a que los procesos realizados por la primera (seguridad) para cada transacción terminan siendo más complejos y requieren mayor cantidad de trabajo por parte del servidor para lograr cumplir la transacción. En este caso se vería reflejado en el porcentaje de consumo del CPU por transacción y el tiempo que demora cada una en realizarse; sin seguridad este consumo y tiempo debería ser menor.





Análisis:

Comparando las gráficas obtenidas del consumo del CPU para el caso con seguridad contra sin seguridad se pudo observar que no se obtuvieron la respuesta esperada. En el caso sin seguridad al comienzo de la transacción siempre se consume un mayor porcentaje de CPU que

en el caso de la aplicación implementando seguridad. Al hacer uso de 2 threads en el caso de la aplicación sin seguridad, el promedio del uso de porcentaje de CPU es mayor al caso con seguridad, contradiciendo así lo esperado en el numeral anterior. Lo mismo ocurre con el uso de 1 thread, sin embargo, esta diferencia de usos no es tan notoria como en el otro caso. Esto puede ocurrir por diversos factores, como la arquitectura del computador en la que se estaba corriendo la aplicación o algún cambio en la conexión que se tenía. Si se analizara también el tiempo que dura cada transacción se podría observar un decremento considerable con respecto al protocolo con seguridad, nuestra hipótesis sería que con nuestro diseño de los monitores de uso de CPU el que la transacción sea más corta podría registrar un uso mayor de CPU. Es decir, el método usado para registrar este valor podría ser sensible a la duración de la transacción.