

**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL
DE HUAMANGA**

FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



TESIS:

**Análisis comparativo del rendimiento en persistencia de
datos entre paradigmas imperativo y reactivo, mediante
aplicaciones CRUD, Ayacucho, 2025**

Para optar el título profesional de:
INGENIERO DE SISTEMAS

PRESENTADO POR:

Bach. Leonel Bradons ROJAS HUARANCCA

ASESOR:

Dr. Manuel Avelino LAGOS BARZOLA

AYACUCHO - PERÚ

2025

Dedicatoria

Dedico a las personas relevantes que me apoyaron en este proceso. En primer lugar, a mis padres, por su apoyo constante y por ser un pilar esencial para alcanzar mi meta de culminar mis estudios universitarios. En segundo lugar, a los docentes quienes con dedicación, experiencia y conocimientos contribuyeron de manera significativa a mi formación profesional.

Agradecimiento

Un agradecimiento a la Universidad Nacional San Cristóbal de Huamanga, Alma Máter, por brindarme la oportunidad de desarrollar y fortalecer mis capacidades y competencias profesionales.

A mi querida Escuela Profesional de Ingeniería de Sistemas, a los docentes, quienes con su compromiso y entrega contribuyeron a mi formación académica para el desempeño en la vida profesional.

Al Dr. Manuel Lagos Barzola, por su valioso apoyo, orientación y guía a lo largo del desarrollo de la presente tesis.

Resumen

En el desarrollo de software, la elección del paradigma de programación tiene un impacto considerable en el rendimiento del sistema, especialmente en operaciones de persistencia de datos como interacciones con bases de datos, almacenamiento en archivos y procesamiento de datos en tiempo real. En la actualidad, los desarrolladores de software se enfrentan al reto de construir aplicaciones cada vez más eficientes, escalables y reactivas. En este contexto, los paradigmas de programación juegan un papel importante en el diseño de soluciones tecnológicas.

El paradigma imperativo, caracterizado por instrucciones secuenciales que modifican explícitamente el estado del programa, es ampliamente utilizado por su simplicidad y compatibilidad con sistemas tradicionales (Skinait, 2023).

Por otro lado, el paradigma reactivo, se centra en qué hacer cuando algo cambia, gestionando flujos de datos y eventos de forma no bloqueante, lo que mejora la escalabilidad y la capacidad de respuesta de las aplicaciones (Jhontona, 2025).

Ante esta situación, surge la necesidad de realizar una investigación comparativa, centrada en la comparación de dos aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo. Para ello se desarrollaron aplicaciones construidas utilizando Java como lenguaje base con el framework spring y sus módulos de spring web y spring webflux.

El propósito de este estudio fue comparar el rendimiento en la persistencia de datos de las dos aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo, mediante un análisis basado en métricas de rendimiento como el tiempo de respuesta, el consumo de recursos del sistema, el throughput y la latencia, evaluadas en distintos escenarios de pruebas de carga, estrés y picos. Esta evaluación permitió identificar, con base en los resultados obtenidos, las diferencias en el comportamiento de ambas aplicaciones CRUD, desarrolladas bajo los paradigmas imperativo y reactivo, respecto al rendimiento en persistencia de datos.

Los resultados descriptivos obtenidos muestran que la aplicación desarrollada bajo el paradigma reactivo presentó menores tiempos de respuesta promedio respecto a la desarrollada bajo el paradigma imperativo en las operaciones Create, Update y Read, no así en la operación Delete, donde el aplicativo imperativo registró un tiempo de respuesta promedio menor. De manera similar, la aplicación reactiva presentó menor latencia

promedio respecto a la aplicación imperativa en los tres endpoints antes mencionados; la operación Delete se excluyó de este análisis, dado que no fue posible registrar dicho indicador al retornar un código de estado HTTP 204 (No Content), el cual no incluye cuerpo de respuesta. En cuanto al consumo de recursos, la aplicación reactiva presentó un mayor porcentaje de uso de CPU y un mayor consumo de memoria tanto del aplicativo como de la base de datos respecto a la aplicación imperativa; sin embargo, presentó menor consumo de memoria de la máquina virtual Java (JVM) respecto a esta última. Asimismo, la aplicación reactiva alcanzó una mayor tasa de transacciones por segundo (TPS) respecto a la aplicación imperativa en la generalidad de las operaciones.

Palabras clave

Paradigma imperativo, paradigma reactivo, spring web, webflux, rendimiento

Abstract

In software development, the choice of programming paradigm has a significant impact on system performance, especially in data persistence operations such as database interactions, file storage, and real-time data processing. Currently, software developers face the challenge of building increasingly efficient, scalable, and responsive applications. In this context, programming paradigms play a crucial role in the design of technological solutions.

The imperative paradigm, characterized by sequential instructions that explicitly modify the program state, is widely used due to its simplicity and compatibility with traditional systems (Skinait, 2023).

On the other hand, the reactive paradigm focuses on what to do when something changes, managing data flows and events in a non-blocking manner, which improves the scalability and responsiveness of applications (Jhontona, 2025).

Given this situation, the need arises to conduct comparative research, focusing on comparing two CRUD applications developed under the imperative and reactive paradigms. To this end, applications were developed using Java as the base language with the Spring framework and its Spring Web and Spring WebFlux modules.

The purpose of this study was to compare the data persistence performance of two CRUD applications developed under imperative and reactive programming paradigms, through an analysis based on performance metrics such as response time, system resource consumption, throughput, and latency, evaluated in different load, stress, and peak test scenarios. This evaluation allowed us to identify, based on the results obtained, the differences in the behavior of both CRUD applications, developed under the imperative and reactive paradigms, with respect to data persistence performance.

The descriptive results obtained show that the application developed under the reactive paradigm exhibited lower average response times compared to the one developed under the imperative paradigm in the Create, Update, and Read operations, but not in the Delete operation, where the imperative application registered a lower average response time. Similarly, the reactive application exhibited lower average latency compared to the imperative application across the three aforementioned endpoints. The Delete operation was excluded from this analysis, as it could not be recorded due to the return of an HTTP status code 204 (No Content), which does not include a response body. Regarding

resource consumption, the reactive application showed a higher percentage of CPU usage and greater memory consumption for both the application and the database compared to the imperative application; however, it exhibited lower Java Virtual Machine (JVM) memory consumption. Furthermore, the reactive application achieved a higher transaction rate per second (TPS) compared to the imperative application across most operations.

Key words

Imperative paradigm, reactive paradigm, Spring Web, Webflux, performance

Índice

Dedicatoria.....	iii
Agradecimiento	iv
Resumen	v
Abstract.....	vii
Índice de figuras	xiv

Capítulo I

Introducción

1.1	Diagnóstico y enunciado del problema.....	17
1.2	Formulación del problema	19
1.2.1	Problema general	19
1.2.2	Problemas específicos.....	19
1.3	Objetivos de la investigación	19
1.3.1	Objetivo general	19
1.3.2	Objetivos específicos	19
1.4	Hipótesis	20
1.4.1	Hipótesis general	20
1.4.2	Hipótesis específicas.....	20
1.5	Justificación	20
1.5.1	Justificación técnica.....	20
1.5.2	Justificación económica.....	21
1.5.3	Justificación social.....	22
1.6	Limitación de la investigación	22

Capítulo II

Marco teórico

2.1	Marco teórico	24
2.1.1	Antecedentes de la investigación.....	24
2.1.2	Paradigma de la programación	26
2.1.3	Programación imperativa.....	26
2.1.4	Programación reactiva	27
2.1.5	Frameworks y tecnologías empleadas	28
2.1.6	Herramientas de evaluación de rendimiento	31
2.2	Marco conceptual.....	32

2.2.1	Programación imperativa.....	32
2.2.2	Programación reactiva	32
2.2.3	Persistencia de datos	32
2.2.4	Tiempo de respuesta	32
2.2.5	Consumo de recursos.....	32
2.2.6	Latencia de operaciones.....	33
2.2.7	Throughput	33
2.2.8	SQL Server	34
2.2.9	Spring Web	34
2.2.10	Spring WebFlux.....	34
2.2.11	JMeter	34
2.2.12	Grafana Lab	34

Capítulo III

Materiales y métodos

3.1	Tipo de investigación.....	35
3.2	Nivel de investigación	35
3.3	Diseño de la investigación	35
3.4	Población y muestra.....	36
3.5	Unidad de análisis.....	37
3.5.1	Aplicaciones desarrolladas	37
3.6	Variable e indicadores	38
3.6.1	Definición conceptual de la variable	38
3.6.2	Definición operacional de la variable.....	39
3.7	Técnicas e instrumentos para el tratamiento de datos e información	40
3.7.1	Técnicas para recolectar información.....	40
3.7.2	Instrumentos para recolectar información	41
3.7.3	Herramientas para el tratamiento de datos e información	41
3.7.4	Procedimiento.....	43
3.7.5	Técnicas para el procesamiento y análisis de datos.....	44

Capítulo IV

Resultados y discusión

4.1	Arquitectura de las aplicaciones CRUD construidas	46
4.2	Código fuente de las aplicaciones.....	47
4.3	Detalles de los endpoints	49

4.4	Tablas de la base de datos	52
4.5	Presentación de los resultados	54
4.5.1	Pruebas de carga	55
4.5.2	Pruebas de estrés.....	86
4.5.3	Pruebas de picos	118
4.6	Análisis comparativo descriptivo entre los 2 aplicativos construidos bajo los paradigmas imperativo y reactivo	148
4.7	Interpretación de resultados	180
4.8	Contrastación descriptiva de hipótesis.....	182
4.9	Discusión del resultado	183

Capítulo V

Conclusiones y recomendaciones

5.1	Conclusiones.....	186
5.2	Recomendaciones	187
	Referencias bibliográficas	189
	Anexos.....	195
	Anexo A: Matriz de consistencia.....	195
	Anexo B: Resultados de los resultados de la prueba de carga del aplicativo sincrónico	201
	Anexo C: Resultados de los resultados de la prueba de estrés del aplicativo sincrónico	202
	Anexo D: Resultados de los resultados de la prueba de picos del aplicativo sincrónico	203
	ANEXO B: Resultados de los resultados de la prueba de carga del aplicativo asincrónico	204
	Anexo C: Resultados de los resultados de la prueba de estrés del aplicativo asincrónico	204
	Anexo D: Resultados de los resultados de la prueba de picos del aplicativo asincrónico	205
	Anexo E: Resultados de throughput en la prueba de carga del aplicativo sincrónico	206
	Anexo F: Resultados de tiempo de respuesta vs usuarios en la prueba de carga del aplicativo sincrónico	207
	Anexo G: Resultados de tiempo de respuesta en la prueba de carga del aplicativo sincrónico.....	208
	Anexo H: Resultados de TPS vs usuarios en la prueba de carga del aplicativo sincrónico.....	209

Anexo I: Resultados de throughput en la prueba de estrés del aplicativo sincrónico	210
Anexo J: Resultados de tiempo de respuesta vs usuarios en la prueba de estrés del aplicativo sincrónico	211
Anexo K: Resultados de tiempo de respuesta en la prueba de estrés del aplicativo sincrónico.....	212
Anexo L: Resultados de TPS vs usuarios en la prueba de estrés del aplicativo sincrónico.....	213
Anexo LL: Resultados de throughput en la prueba de picos del aplicativo sincrónico	214
Anexo M: Resultados de tiempo de respuesta vs usuarios en la prueba de picos del aplicativo sincrónico	215
Anexo N: Resultados de tiempo de respuesta en la prueba de picos del aplicativo sincrónico.....	216
Anexo Ñ: Resultados de TPS vs usuarios en la prueba de picos del aplicativo sincrónico.....	217
Anexo O: Resultados de throughput en la prueba de carga del aplicativo asincrónico	218
Anexo P: Resultados de tiempo de respuesta vs usuarios en la prueba de carga del aplicativo asincrónico	219
Anexo Q: Resultados de tiempo de respuesta en la prueba de carga del aplicativo asincrónico	220
Anexo R: Resultados de TPS vs usuarios en la prueba de carga del aplicativo asincrónico	221
Anexo S: Resultados de throughput en la prueba de estrés del aplicativo asincrónico	222
Anexo T: Resultados de tiempo de respuesta vs usuarios en la prueba de estrés del aplicativo asincrónico	223
Anexo U: Resultados de tiempo de respuesta en la prueba de estrés del aplicativo asincrónico	224
Anexo V: Resultados de TPS vs usuarios en la prueba de estrés del aplicativo asincrónico	225
Anexo W: Resultados de throughput en la prueba de picos del aplicativo asincrónico	226
Anexo X: Resultados de tiempo de respuesta vs usuarios en la prueba de estrés del aplicativo asincrónico	227
Anexo Y: Resultados de tiempo de respuesta en la prueba de picos del aplicativo asincrónico	228
Anexo Z: Resultados de TPS vs usuarios en la prueba de picos del aplicativo asincrónico	229

Índice de tablas

Tabla 1	Herramientas para el tratamiento de datos	41
Tabla 2	Herramientas de recolección y análisis de datos	42
Tabla 3	Detalle del servicio POST para la creación de data.	50
Tabla 4	Detalle del servicio GET para la obtención de data.	51
Tabla 5	Detalle del servicio PUT para la edición de data.	51
Tabla 6	Detalle del servicio DELETE para la eliminación de data.....	52
Tabla 7	Resultados del aplicativo sincrónico en la prueba de carga	56
Tabla 8	Resultados del aplicativo asincrónico en la prueba de carga	72
Tabla 9	Resultados del aplicativo sincrónico en la prueba de estrés.....	87
Tabla 10	Resultados del aplicativo asincrónico en la prueba de estrés	103
Tabla 11	Resultados del aplicativo sincrónico en la prueba de picos	119
Tabla 12	Resultados del aplicativo asincrónico en la prueba de picos.....	134
Tabla 13	Tabla de datos de tiempo de respuesta para la prueba de carga	150
Tabla 14	Tabla comparativa de tiempo de respuesta para la prueba de carga.....	150
Tabla 15	Tabla de datos de tiempo de respuesta para la prueba de estrés.....	151
Tabla 16	Tabla comparativa de tiempo de respuesta para la prueba de estrés	152
Tabla 17	Tabla de datos de tiempo de respuesta para la prueba de picos	152
Tabla 18	Tabla comparativa de tiempo de respuesta para la prueba de picos.....	153
Tabla 19	Tabla de datos de tiempo de consumo de recursos para la prueba de carga	157
Tabla 20	Tabla comparativa de tiempo de consumo de recursos para la prueba de carga	158
Tabla 21	Tabla de datos de tiempo de consumo de recursos para la prueba de estrés	159
Tabla 22	Tabla comparativa de tiempo de consumo de recursos para la prueba de estrés	160
Tabla 23	Tabla de datos de tiempo de consumo de recursos para la prueba de picos.	160
Tabla 24	Tabla comparativa de tiempo de consumo de recursos para la prueba de picos	161
Tabla 25	Tabla de datos de latencia para la prueba de carga	167
Tabla 26	Tabla comparativa de Latencia para la prueba de carga.....	167
Tabla 27	Tabla de datos de latencia para la prueba de estrés	168
Tabla 28	Tabla comparativa de Latencia para la prueba de estrés	169
Tabla 29	Tabla de datos de latencia para la prueba de picos.....	169
Tabla 30	Tabla comparativa de Latencia para la prueba de picos.....	170
Tabla 31	Tabla de datos de throughput para la prueba de carga	174
Tabla 32	Tabla comparativa de throughput para la prueba de carga.....	174
Tabla 33	Tabla de datos de throughput para la prueba de estrés.....	175
Tabla 34	Tabla comparativa de throughput para la prueba de estrés	176
Tabla 35	Tabla de datos de throughput para la prueba de picos.....	176
Tabla 36	Tabla comparativa de throughput para la prueba de picos	177

Índice de figuras

Figura 1 Modelo de intercambio de información a través de una API.....	30
Figura 2 Arquitectura de las aplicaciones CRUD	46
Figura 3 Prueba de carga	55
Figura 4 Gráfica de latencia del aplicativo sincrónico	58
Figura 5 Gráfica de tiempo de respuesta durante la prueba	59
Figura 6 Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico	60
Figura 7 Gráfica de distribución de tiempo de respuesta	60
Figura 8 Gráfica de TPS en general del aplicativo sincrónico	61
Figura 9 Gráfica del TPS por operación del aplicativo sincrónico	62
Figura 10 Gráfica del consumo de CPU del aplicativo sincrónico.....	63
Figura 11 Gráfica de consumo de memoria del aplicativo sincrónico	63
Figura 12 Gráfica de consumo de JVM Heap Memory del aplicativo sincrónico	64
Figura 13 Gráfica de consumo de JVM Pause Time del aplicativo sincrónico.....	64
Figura 14 Gráfica de consumo de JVM Threads del aplicativo sincrónico.....	65
Figura 15 Gráfica de conexiones de HikariCP del aplicativo sincrónico.....	66
Figura 16 Gráfica de tiempo de espera de conexión del aplicativo sincrónico	67
Figura 17 Gráfica de throughput del consumo de sql server del aplicativo sincrónico..	67
Figura 18 Gráfica de lock y deadlock del aplicativo sincrónico	68
Figura 19 Gráfica de latencia del disco del aplicativo sincrónico	69
Figura 20 Gráfica de percentiles del aplicativo sincrónico	69
Figura 21 Gráfica de latencia del aplicativo asincrónico	74
Figura 22 Gráfica de tiempo de respuesta durante la prueba	75
Figura 23 Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico	76
Figura 24 Gráfica de distribución de tiempo de respuesta	76
Figura 25 Gráfica de TPS en general del aplicativo asincrónico	77
Figura 26 Gráfica del TPS por operación del aplicativo asincrónico.....	78
Figura 27 Gráfica del consumo de CPU del aplicativo asincrónico.....	78
Figura 28 Gráfica de consumo de memoria del aplicativo asincrónico	79
Figura 29 Gráfica de JVM Heap Memory del aplicativo asincrónico.....	79
Figura 30 Gráfica de JVM GC Pause Time del aplicativo asincrónico.....	80
Figura 31 Gráfica de JVM Threads del aplicativo asincrónico	81
Figura 32 Gráfica de conexiones de R2dbc del aplicativo asincrónico.....	82
Figura 33 Gráfica de throughput del consumo de sql server del aplicativo asincrónico	82
Figura 34 Gráfica de lock y deadlock del aplicativo asincrónico.....	83
Figura 35 Gráfica de latencia del disco del aplicativo asincrónico	83
Figura 36 Gráfica de percentiles del aplicativo asincrónico.....	84
Figura 37 Prueba de estrés.....	87
Figura 38 Gráfica de latencia del aplicativo sincrónico	89
Figura 39 Gráfica de tiempo de respuesta durante la prueba	91
Figura 40 Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico	91

Figura 41	Gráfica de distribución de tiempo de respuesta	92
Figura 42	Gráfica de TPS en general del aplicativo sincrónico	93
Figura 43	Gráfica del TPS por operación del aplicativo sincrónico.....	94
Figura 44	Gráfica del consumo de CPU del aplicativo sincrónico.....	94
Figura 45	Gráfica de consumo de memoria del aplicativo sincrónico	95
Figura 46	Gráfica de consumo de JVM Heap Memory del aplicativo sincrónico	96
Figura 47	Gráfica de consumo de JVM del aplicativo sincrónico	96
Figura 48	Gráfica de consumo de JVM Threads del aplicativo sincrónico.....	97
Figura 49	Gráfica de conexiones de HikariCP del aplicativo sincrónico.....	98
Figura 50	Gráfica de tiempo de espera de conexión del aplicativo sincrónico	99
Figura 51	Gráfica de throughput del consumo de SQL server del aplicativo sincrónico	99
Figura 52	Gráfica de lock y deadlock del aplicativo sincrónico	100
Figura 53	Gráfica de latencia del disco del aplicativo sincrónico	100
Figura 54	Gráfica de percentiles del aplicativo sincrónico	101
Figura 55	Gráfica de latencia del aplicativo asincrónico	105
Figura 56	Gráfica de tiempo de respuesta durante la prueba	107
Figura 57	Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo asincrónico.....	107
Figura 58	Gráfica de distribución de tiempo de respuesta	108
Figura 59	Gráfica de TPS en general del aplicativo asincrónico	108
Figura 60	Gráfica del TPS por operación del aplicativo asincrónico.....	109
Figura 61	Gráfica del consumo de CPU del aplicativo asincrónico.....	110
Figura 62	Gráfica de consumo de memoria del aplicativo asincrónico	110
Figura 63	Gráfica de consumo de JVM Heap Memory del aplicativo asincrónico	111
Figura 64	Gráfica de consumo de JVM GC Pause Time del aplicativo asincrónico ..	112
Figura 65	Gráfica de consumo de JVM Threads del aplicativo asincrónico.....	112
Figura 66	Gráfica de conexiones de R2dbc del aplicativo asincrónico.....	113
Figura 67	Gráfica de throughput del consumo de SQL server del aplicativo asincrónico	114
Figura 68	Gráfica de lock y deadlock del aplicativo asincrónico.....	114
Figura 69	Gráfica de latencia del disco del aplicativo asincrónico	115
Figura 70	Gráfica de percentiles del aplicativo asincrónico.....	116
Figura 71	Prueba de picos	118
Figura 72	Gráfica de latencia del aplicativo sincrónico	120
Figura 73	Gráfica de tiempo de respuesta durante la prueba	122
Figura 74	Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico	122
Figura 75	Gráfica de distribución de tiempo de respuesta	123
Figura 76	Gráfica de TPS en general del aplicativo sincrónico	123
Figura 77	Gráfica del TPS por operación del aplicativo sincrónico.....	124
Figura 78	Gráfica del consumo de CPU del aplicativo sincrónico.....	125
Figura 79	Gráfica de consumo de memoria del aplicativo sincrónico	125
Figura 80	Gráfica de consumo de JVM Heap Memory del aplicativo sincrónico	126

Figura 81	Gráfica de consumo de JVM GC Pause Time del aplicativo sincrónico	127
Figura 82	Gráfica de consumo de JVM Threads del aplicativo sincrónico.....	127
Figura 83	Gráfica de conexiones de HikariCP del aplicativo sincrónico.....	128
Figura 84	Gráfica de tiempo de espera de conexión del aplicativo sincrónico	129
Figura 85	Gráfica de throughput del consumo de SQL server del aplicativo sincrónico	129
Figura 86	Gráfica de lock y deadlock del aplicativo sincrónico	130
Figura 87	Gráfica de latencia del disco del aplicativo sincrónico	131
Figura 88	Gráfica de percentiles del aplicativo sincrónico	131
Figura 89	Gráfica de latencia del aplicativo asincrónico	136
Figura 90	Gráfica de tiempo de respuesta durante la prueba	137
Figura 91	Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo asincrónico.....	137
Figura 92	Gráfica de distribución de tiempo de respuesta	138
Figura 93	Gráfica de TPS en general del aplicativo asincrónico	139
Figura 94	Gráfica del TPS por operación del aplicativo asincrónico.....	139
Figura 95	Gráfica del consumo de CPU del aplicativo asincrónico.....	140
Figura 96	Gráfica de consumo de memoria del aplicativo asincrónico	141
Figura 97	Gráfica de consumo de JVM Heap Memory del aplicativo asincrónico	141
Figura 98	Gráfica de consumo de JVM GC Pause Time del aplicativo asincrónico ..	142
Figura 99	Gráfica de consumo de JVM Threads del aplicativo asincrónico.....	143
Figura 100	Gráfica de conexiones de R2dbc del aplicativo asincrónico.....	143
Figura 101	Gráfica de throughput del consumo de SQL server del aplicativo asincrónico.....	144
Figura 102	Gráfica de lock y deadlock del aplicativo asincrónico.....	145
Figura 103	Gráfica de latencia del disco del aplicativo asincrónico	145
Figura 104	Gráfica de percentiles del aplicativo asincrónico.....	146

Capítulo I

Introducción

1.1 Diagnóstico y enunciado del problema

En el desarrollo de software, la elección del paradigma de programación tiene un impacto significativo en el rendimiento del sistema, especialmente en operaciones de persistencia de datos como interacciones con bases de datos, almacenamiento en archivos y procesamiento de datos en tiempo real. En la actualidad, los desarrolladores de software se enfrentan al reto de construir aplicaciones cada vez más eficientes, escalables y reactivas. En este contexto, los paradigmas de programación juegan un papel determinante en el diseño de soluciones tecnológicas.

El paradigma imperativo, se fundamenta en la ejecución secuencial de instrucciones y el control explícito del estado del sistema. Este paradigma ha demostrado ser eficiente en aplicaciones de complejidad moderada y en contextos donde la concurrencia no representa un requisito crítico, organizaciones como Basecamp han defendido arquitecturas monolíticas y modelos más tradicionales, argumentando que la simplicidad estructural mejora la mantenibilidad y reduce la complejidad innecesaria (Fried & Hansson, 2018).

No obstante, el crecimiento de aplicaciones distribuidas, servicios en la nube y sistemas de alta demanda ha impulsado la adopción del paradigma reactivo, el cual se basa en flujos de datos asíncronos y en la propagación automática de eventos. Un caso representativo es Netflix, que implementó principios de programación reactiva y herramientas como ReactiveX para gestionar millones de solicitudes concurrentes con alta resiliencia y baja latencia (Netflix Technology Blog, 2013).

Los ejemplos mencionados anteriormente, como Netflix y Basecamp, representan práctica del uso y la adopción de los paradigmas imperativo y reactivo. A continuación, se presentan estudios e investigaciones científicas que analizan empíricamente las ventajas, limitaciones y desempeño de cada paradigma, ofreciendo un respaldo académico y comparativo.

El paradigma imperativo, caracterizado por instrucciones secuenciales que modifican explícitamente el estado del programa, es ampliamente utilizado por su simplicidad y compatibilidad con sistemas tradicionales (Skinait, 2023).

Según Calzado Perez (2020) señala que las operaciones de persistencia estructurada en bases de datos relacionales se benefician de un paradigma imperativo debido a su compatibilidad con frameworks estándares y la facilidad de integración con sistemas existentes.

Otros hallazgos como Espinoza Rojas (2022) concluye que, en tareas con baja concurrencia, el paradigma síncrono (imperativo) puede superar al reactivo en términos de latencia, debido a la menor sobrecarga de abstracciones reactivas.

Estudios recientes, según Karol Lis y Jakub Smolka (2024) han identificado ventajas en cuanto a la capacidad de respuesta del paradigma reactivo, pero también un mayor uso de la CPU en contextos de carga media y alta. Sin embargo, también hubo escenarios en los que la aplicación imperativa estaba mejorando.

Por otra parte, Gamze Koksall. (2024) concluye que la programación reactiva demuestra ventajas sustanciales sobre la programación imperativa tradicional para mejorar la capacidad de respuesta, la eficiencia y rendimiento de los recursos y la escalabilidad de los microservicios.

El paradigma reactivo, se centra en qué hacer cuando algo cambia, gestionando flujos de datos y eventos de forma no bloqueante, lo que mejora la escalabilidad y la capacidad de respuesta de las aplicaciones (Jhontona, 2025).

A partir de la evidencia revisada, se observa que los hallazgos sobre el desempeño de los paradigmas imperativo y reactivo son contradictorios. Mientras algunos estudios destacan las ventajas del paradigma reactivo en capacidad de respuesta, escalabilidad, rendimiento y eficiencia de recursos, otros muestran que el paradigma imperativo puede superar al reactivo en latencia y consumo de recursos en escenarios de baja y moderada concurrencia.

Ante la falta de evidencia comparativa que evalúe aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo, manteniendo las mismas condiciones de infraestructura, configuración tecnológica y criterios de medición, se evidencia una carencia de estudios en el contexto local que permitan analizar las

diferencias de rendimiento en la persistencia de datos en un entorno Java. En ese contexto, surge la necesidad de desarrollar un estudio comparativo que genere evidencia objetiva.

1.2 Formulación del problema

1.2.1 Problema general

¿Cuáles son las diferencias en el rendimiento de la persistencia de datos entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en un entorno Java?

1.2.2 Problemas específicos

- A. ¿Qué diferencias existen en el tiempo de respuesta entre aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo durante las operaciones de persistencia de datos?
- B. ¿Qué diferencias existen en el consumo de recursos del sistema (CPU Y JVM) entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo?
- C. ¿Qué diferencias existen en la latencia de operación entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en los escenarios de pruebas de carga, estrés y picos?
- D. ¿Qué diferencias existen en el throughput alcanzado por las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en los escenarios de pruebas de carga, estrés y picos?

1.3 Objetivos de la investigación

1.3.1 Objetivo general

Comparar el rendimiento en persistencia de datos en las aplicaciones CRUD desarrolladas bajo los paradigmas de programación, imperativo y reactivo, mediante escenarios de pruebas de carga, estrés y picos.

1.3.2 Objetivos específicos

- A. Comparar el tiempo de respuesta de las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo durante las operaciones de persistencia de datos.

- B. Comparar el consumo de recursos del sistema (CPU Y JVM) de las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.
- C. Comparar la latencia en operación de las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo bajo distintos escenarios de prueba de rendimiento.
- D. Medir y comparar el throughput alcanzado por las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo durante las operaciones CRUD.

1.4 Hipótesis

1.4.1 Hipótesis general

Existen diferencias en el rendimiento de la persistencia de datos entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo, en los escenarios de pruebas de carga, estrés y picos.

1.4.2 Hipótesis específicas

- a) Existen diferencias en el tiempo de respuesta promedio entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.
- b) Existen diferencias en el consumo de recursos del sistema (CPU y memoria JVM) entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.
- c) Existen diferencias en la latencia promedio entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.
- d) Existen diferencias en el throughput alcanzado por las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.

1.5 Justificación

1.5.1 Justificación técnica

La justificación técnica de esta investigación radica en la necesidad de evaluar y comparar el comportamiento de los paradigmas de programación imperativo y reactivo en aplicaciones CRUD, considerando aspectos como el tiempo de respuesta, el consumo de recursos y la capacidad de procesamiento, con el propósito de generar evidencia técnica que contribuya a la toma de decisiones en el desarrollo de software. La programación

reactiva permite manejar flujos de datos asíncronos, lo que reduce la latencia y mejora la escalabilidad en sistemas distribuidos, especialmente en entornos de alta concurrencia. El estudio de la Universidad Politécnica de Madrid comparó APIs reactivas y síncronas en Spring-Boot, encontrando que las reactivas manejan mejor el estrés del sistema bajo cargas concurrentes (Vallejo Fernández, 2022).

Por otro lado, la programación imperativa ofrece un control granular sobre las operaciones, lo que puede ser ventajoso en escenarios donde la simplicidad y la predictibilidad son prioritarias (Skinait, 2023).

Sin embargo, los estudios previos se hacen realizado en contextos generales y no abordan escenarios específicos como operaciones CRUD de persistencia de datos, evaluando mediante métricas de rendimiento como el tiempo de respuesta, consumo de recurso, el throughput y la latencia, por ello resulta pertinente realizar un análisis comparativo que permita generar evidencia empírica. La evaluación de las aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo permitirá identificar sus características, fortalezas y limitaciones, proporcionando información objetiva que contribuya a la selección del paradigma más adecuado para aplicaciones orientadas a la persistencia de datos.

1.5.2 Justificación económica

Desde el punto de vista económico, la elección del paradigma de programación impacta directamente los costos de desarrollo, mantenimiento y escalabilidad de las aplicaciones. En entornos de la nube, donde el costo se calcula según el consumo de recursos, una mejora en rendimiento del 20% de CPU y memoria puede reducir los costos de infraestructura entre 500 a 100 USD por cada servidor. (Datamation, 2025)

La programación reactiva puede reducir los costos operativos al mejorar el rendimiento y eficiencia en el uso de recursos computacionales, como CPU y memoria, especialmente en entornos de nube donde los costos están ligados al consumo de recursos (Aguilar Encarnación, 2024). Sin embargo, su curva de aprendizaje más pronunciada puede incrementar los costos iniciales de capacitación y desarrollo.

Por otro lado, las aplicaciones imperativas tienden a ser más fáciles de depurar y mantener debido a su flujo secuencial, lo que disminuye los costos de soporte técnico a largo plazo,

los proyectos que priorizan claridad y predictibilidad en la codificación pueden reducir hasta un 15 % anual en gastos de mantenimiento. (Gartner, 2023)

Este estudio comparativo permitirá en el apoyo de toma de decisiones para la inversión en tecnología, respecto a consumo de recursos (CPU y JVM) como en la inversión de capacitación de sus empleados ya que al seleccionar el paradigma correcto ofrece el mejor balance entre rendimiento y costos.

1.5.3 Justificación social

La elección de un paradigma de programación eficiente tiene implicaciones sociales significativas, especialmente en el contexto de aplicaciones que impactan a comunidades amplias, como sistemas de salud, financiera, educación y servicios públicos. Ya que estos sectores tienen una alta concurrencia de solicitudes para la obtención de información de la persona, por lo cual es fundamental garantizar la información para así no afectar a los procesos de cada flujo y a la experiencia del usuario.

Un mejor rendimiento en la persistencia de datos puede traducirse en aplicaciones más rápidas y confiables, mejorando la experiencia del usuario y la accesibilidad a servicios críticos. Por ejemplo, en sistemas de salud en línea, la rapidez en la persistencia de datos puede ser crucial para procesar información en tiempo real (Pardo Calvache, 2009).

En el sector financiero, la rapidez en el procesamiento de transacciones y la confiabilidad de los datos contribuyen directamente a la experiencia y satisfacción del cliente en la competitividad de los servicios digitales bancarios (Alhasan et al., 2025).

En este estudio comparativo permitirá determinar de manera cuantitativa el tiempo de respuesta como también la cantidad de peticiones por segundos (TPS), para identificar cual paradigma soportan una mayor carga de peticiones simultaneas, lo que es crítico en sistemas de alta concurrencia.

1.6 Limitación de la investigación

En el desarrollo de la investigación se establecen las siguientes limitaciones:

- a) Los resultados se basan en aplicaciones desarrolladas en Java/Spring con SQL Server.
- b) La prueba de carga, estrés y picos se realizaron en un entorno local, no en un servidor ni nube.

- c) Limitación en operaciones CRUD estándar, lo que podría no reflejar en aplicaciones con lógica de negocio más compleja.
- d) Estudio realizado con la versión de Spring 2.7.18, por lo que los resultados podrían variar al utilizar versiones anteriores o posteriores.
- e) Pool de conexiones de ambos aplicativos CRUD mínimas de 8 a máximo de 15.
- f) Memoria Ram asignada a SQL Server de 4GB y 2.5 de CPU y para los aplicativos CRUD 2GB y 1.5 CPU
- g) Los resultados y conclusiones de la presente investigación se limitan al contexto evaluado, en aplicaciones CRUD desarrolladas en Java con los frameworks Spring Web y Spring WebFlux, utilizando SQL Server como base de datos. Por lo tanto, las conclusiones no pretenden generalizarse a otras arquitecturas, lenguajes de programación, frameworks o motores de base de datos distintos a los aquí analizados.
- h) Otra limitación, se presentó una dependencia secuencial entre endpoints: Get requiere que Create haya devuelto un código de estado 200, Update requiere la ejecución exitosa de Get, y Delete requiere que Update se haya ejecutado correctamente. Esta condición pudo influir en los resultados obtenidos.

Capítulo II

Marco teórico

2 Marco teórico y marco conceptual

2.1 Marco teórico

2.1.1 Antecedentes de la investigación

Iwanowski y Kozieł (2022), en su artículo científico realizaron una comparación en aplicaciones web construidas bajo ambos paradigmas, evaluando el tiempo de respuesta, uso de CPU, memoria y estabilidad. Dando como resultados que las aplicaciones reactivas resultaron ser más rápidas que las imperativas en mayoría de los escenarios realizados en la investigación, en cuanto al consumo de recurso de la memoria RAM, la aplicación reactiva utilizó menos CPU (28.9%) en comparación con la imperativa (43.2%). Sin embargo, el consumo de memoria y la estabilidad general fueron similares en ambos. Este antecedente es relevante para la investigación porque demuestra que el paradigma reactivo es eficiente en operaciones concurrentes. No obstante, el estudio no analiza operaciones CRUD con SQL Server.

Vallejo Fernández (2022), en su tesis de grado tuvo como objetivo comparar el desempeño de aplicaciones desarrolladas bajo programación reactiva y programación síncrona utilizando Spring Boot. La metodología consistió en la implementación de dos APIs con funcionalidades equivalentes, sometidas a pruebas de carga y estrés para medir tiempos de respuesta, uso de CPU y estabilidad bajo concurrencia. Los resultados mostraron que el modelo reactivo presentó mejor comportamiento bajo alta concurrencia, manteniendo mayor estabilidad en el tiempo de respuesta, aunque con mayor complejidad de implementación.

Este estudio es relevante para la presente investigación por que evidencia que el paradigma reactivo ofrece ventajas en escenarios de alta carga, lo justifica analizar el comportamiento con la base de datos SQL Server.

Según Kacper Mochnej y Marcin Badurowicz (2023), en su artículo científico realizaron un estudio comparativo sobre el rendimiento de micro servicios desarrollados bajo varios paradigmas, el objetivo fue evaluar el desempeño de ambos paradigmas, mediante pruebas de rendimiento con la herramienta Gatling, y comparar tiempo de respuesta en eficiencia general. Se concluyeron que un procesamiento asíncrono, no siempre ofrecen

mejor rendimiento, especialmente en tareas intensivas en CPU. Aunque mostraron ventajas en operaciones no bloqueantes y menor uso de RAM, en general, los servicios imperativos fueron más eficientes en operaciones menos complejas. Lo que justifica realizar un análisis comparativo en operaciones CRUD.

Nordlund y Nordström (2022), en su tesis de grado realizaron una comparación práctica entre aplicaciones desarrolladas con paradigmas reactivo y no reactivo/imperativo. La metodología consistió en implementar prototipos equivalentes y medir latencia, uso de CPU y uso de memoria bajo diferentes niveles de concurrencia. El análisis mostró que, aunque las aplicaciones reactivas se desempeñaron mejor en algunos escenarios de alta concurrencia, el paradigma no reactivo (imperativo) superó en rendimiento a la implementación WebFlux en varias categorías, con diferencias de entre 10 % y 20 % en tiempo de respuesta y uso de recursos medidos bajo cargas específicas.

Este antecedente es relevante porque indica que una arquitectura tradicional imperativa puede tener mejor desempeño en situaciones reales dependiendo del tipo de carga y la configuración del sistema, respaldando la necesidad de un análisis específico para operaciones CRUD con SQL Server.

A nivel nacional, existen trabajos que abordan construcciones de software mediante la programación reactiva y programación sincrónica, por ejemplo, Lozada Paredes y Valle Alvarado (2026), en su tesis de grado implementó un sistema web para la solicitud de factoring electrónico en una institución bancaria privada, utilizando programación reactiva y una arquitectura basada en la nube. La investigación evidenció que la adopción de flujos asíncronos y componentes cloud permitió una reducción del 70 % en los tiempos de procesamiento, optimizando significativamente la eficiencia del sistema y la experiencia de los usuarios.

Como también las construcciones de software, que normalmente lo realizan de manera tradicional de manera imperativa, por ejemplo, Arevalo Diaz y Alvarado Estanga (2024) en su tesis de grado desarrollaron un sistema de manera tradicional de manera monolítica para generar reportes de pruebas automatizadas en aplicaciones web utilizando RPA, con el objetivo de mejorar la eficiencia de la fase de pruebas dentro de metodologías ágiles. Mediante indicadores se evaluó la eficacia y rendimiento del proceso de pruebas.

Ante la ausencia de evidencia empírica en el contexto local que compare aplicaciones CRUD desarrolladas en java bajo los paradigmas de programación imperativo y reactivo,

manteniendo las mismas condiciones tecnológicas y criterios de evaluación, así como la escasez de estudios orientados a operaciones de persistencia de datos, surge la necesidad de realizar un análisis comparativo que permita evaluar el comportamiento de ambas aplicaciones mediante métricas de tiempo de respuesta, latencia, throughput y consumo de recursos del sistema.

2.1.2 Paradigma de la programación

De acuerdo con Wirth (1971), un paradigma de programación define un conjunto de conceptos, métodos y prácticas que guían la forma en que se estructura y desarrolla un programa, determinando así cómo se resuelven problemas dentro del marco tecnológico.

En este sentido, Ario Liyan (2023) afirma que un paradigma de programación es una forma estructurada y conceptual para crear software. Representa un conjunto de principios, modelos y técnicas que guían lo que se diseña, organiza y escribe el código. Estos paradigmas influyen directamente en la forma de pensar del programador, en la solución de problemas y en la manera de procesar cálculos y algoritmos.

2.1.3 Programación imperativa

Según Quintanar Perez, David (2024), “La programación imperativa se centra en describir paso a paso cómo realizar una tarea. En este enfoque, un programa se compone de una serie de instrucciones que modifican el estado del programa a lo largo del tiempo, es como darle a la computadora una lista detallada de tareas a realizar, una tras otra.” Compuesta de principios fundamentales los cuales son:

Secuencia de instrucciones:

Son instrucciones que se ejecutan una tras otra, en el orden en que fueron escritas. Esto significa que cambiar ese orden puede cambiar completamente lo que hace el programa.

Estado mutable:

Se trabaja con variables y estructuras que pueden cambiar a lo largo del tiempo. Esto permite ir guardando y modificando información según lo que va ocurriendo en el programa.

Estructuras de control:

Usar condicionales, bucles y funciones es clave, estas herramientas permiten que el programa tome decisiones, repita acciones o agrupe instrucciones de forma más organizada.

Asimismo, Ernest Rudnicki (2025), afirma que la programación imperativa implica "escribir programas como secuencias de comandos explícitos que se ejecutan en orden de arriba a abajo", donde el programador especifica los pasos exactos que la computadora debe tomar, utilizando asignaciones de variables, bucles y lógica condicional para controlar el flujo y manipular datos.

De manera similar Alemana Cocca (2022) afirma que la programación imperativa se basa en dar instrucciones precisas a la computadora, indicando paso a paso cómo debe ejecutarse un programa. Se llama imperativa porque el programador le dice exactamente qué hacer y en qué orden.

2.1.4 Programación reactiva

Kailash Nirmal (2025), define que la programación reactiva es un paradigma que permite desarrollar aplicaciones capaces de responder de forma rápida y eficiente a los cambios en los datos. En lugar de seguir un modelo tradicional de solicitud-respuesta, se trata de flujos de datos asíncronos y en la propagación de eventos, lo que facilita la creación de sistemas receptivos, resilientes y escalables. Este enfoque es útilmente en contextos donde se manejan múltiples usuarios o eventos en tiempo real, como aplicaciones web, microservicios o procesamiento de datos. Los principios involucrados en este paradigma son:

El patrón de observador, que permite que varios componentes se suscriban a los datos y reaccionen automáticamente cuando ocurren cambios, favoreciendo una arquitectura desacoplada.

La contrapresión es otra característica importante: regula el flujo de datos entre productores y consumidores, evitando que el sistema se sobrecargue si no puede procesar la información lo suficientemente rápido.

La composición funcional, permitiendo encadenar operaciones como filtrado, mapeo y transformación de datos, lo que hace que el código sea más limpio, modular y fácil de mantener.

Del mismo modo Rahul Krishnan (2024), aporta que la programación reactiva es un paradigma centrado en flujos de datos y en responder a los cambios en tiempo real. En lugar de verificar constantemente si hay alguna respuesta (como en métodos tradicionales), el código se suscribe a eventos y reacciona solo cuando ocurren, lo que mejora el rendimiento y ahorra recursos. En Java, esto se traduce en trabajar con flujos asíncronos ofrece mayor flexibilidad y eficiencia gracias a su enfoque declarativo y más sencillo para manejar conexiones y errores.

2.1.5 Frameworks y tecnologías empleadas

2.1.5.1 IntelliJ IDEA

Para JetBrains, es un IDE diseñado para maximizar la productividad del desarrollador. Se enfoca en proporcionar una experiencia de desarrollo integral que cubre todos los aspectos del ciclo de vida del software, desde la codificación hasta la depuración, el refactoring, el análisis de código y la integración con herramientas de construcción y control de versiones. Promocionan características como el completado de código sensible, la refactorización segura y un potente conjunto de herramientas integradas para mejorar la calidad del código y la velocidad de desarrollo.

2.1.5.2 Spring

La documentación oficial de Spring Framework define a Spring como un "modelo de programación y configuración completo para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de despliegue". Spring se enfoca en la "infraestructura" de las aplicaciones empresariales, permitiendo que los equipos se concentren en la lógica de negocio sin ataduras innecesarias a entornos de despliegue específicos.

2.1.5.3 SQL Server

Según Coronel y Morris (2018): "SQL Server es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado por Microsoft. Permite crear, administrar y consultar bases de datos mediante el uso de SQL. Está diseñado para facilitar la escalabilidad, la seguridad y la integridad de los datos. Su arquitectura cliente-servidor permite aplicaciones distribuidas con altos niveles de rendimiento".

Por otro lado, Microsoft Learn (2024) define que SQL Server es un SGBD relacional desarrollado por Microsoft que ofrece una plataforma robusta y segura para la gestión de

datos empresariales. Incluye funciones como integración con .NET, soporte para procedimientos almacenados en T-SQL, conectividad con Azure Synapse Analytics y herramientas como Data Tools, Integration Services (SSIS), Analysis Services (SSAS) y Reporting Services (SSRS).

2.1.5.4 Spring Web

Según Johnson, Hoeller y Arendsen (2022) define que Spring Web es un módulo del framework Spring que proporciona soporte para el desarrollo de aplicaciones web basado en el patrón MVC (Modelo-Vista-Controlador). En la cual integra con tecnologías como Thymeleaf, JSP o REST para construir aplicaciones orientadas a servicios y arquitecturas modulares. Ofrece mecanismos para el manejo de peticiones HTTP, validación, enlace de formularios, e integración con servicios web.

Por otro lado, Walls (2016) lo define como un estándar en el desarrollo de aplicaciones web empresariales gracias a su enfoque ligero y desacoplado. La arquitectura de Spring Web permite separar claramente la lógica de negocio de la lógica de presentación, facilitando la mantenibilidad del código.

2.1.5.5 Spring WebFlux

Según Hoeller y Johnson (2022) define como un módulo reactivo del framework Spring diseñado para manejar cargas concurrentes y asíncronas de manera eficiente. Utiliza el modelo de programación basado en flujos reactivos (Reactive Streams) e incorpora bibliotecas como Reactor para permitir el desarrollo de aplicaciones web altamente escalables.

Por otro lado, Walls (2018), Spring WebFlux está diseñado para aplicaciones donde el rendimiento y la escalabilidad son fundamentales. WebFlux no se basa en el modelo tradicional bloqueante de servlet, sino que adopta un enfoque completamente no bloqueante y basado en eventos, lo cual es bueno para arquitecturas reactivas y microservicios.

2.1.5.6 API Rest

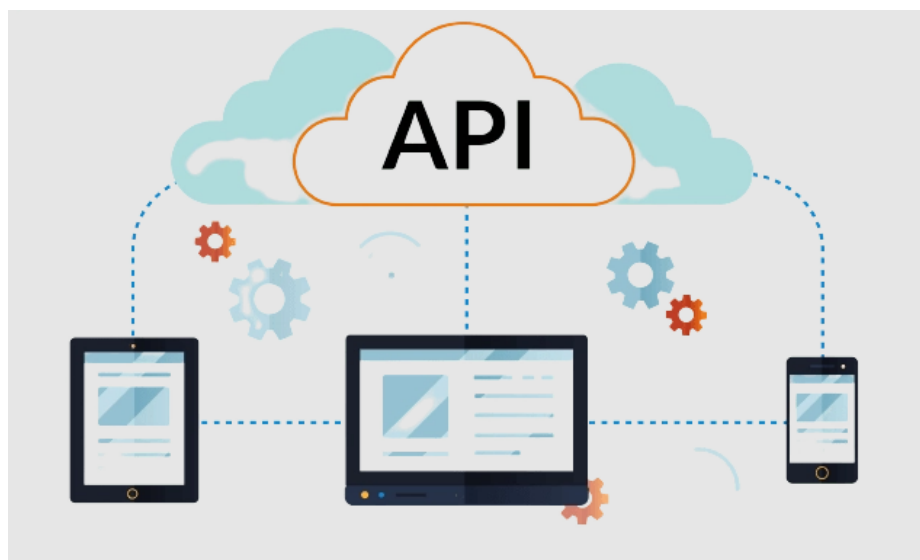
Según Jain V. K. describe REST como una "arquitectura que permite la comunicación y el intercambio de información entre sistemas distribuidos" a través de HTTP. REST facilita la integración de diferentes servicios y plataformas gracias a su simplicidad y al uso de estándares web bien establecidos.

Para Red Hat (2020) describe las API REST como "APIs que se adhieren a los principios de diseño de REST". Resaltan que estas APIs permiten la interacción entre sistemas a través de HTTP (Figura 1), de una manera flexible y escalable. La perspectiva de los Rest enfatiza la simplicidad y la capacidad para conectar diversos componentes de software de manera eficiente, lo que las hace ideales para microservicios y aplicaciones modernas.

Por otro lado, IBM Developer (2025) describe una API REST como un conjunto de funciones que los desarrolladores pueden utilizar para solicitar servicios desde otro sistema. Enfatizan que las API REST utilizan el protocolo HTTP para las comunicaciones y que se adhieren a los principios arquitectónicos de REST, permitiendo la comunicación entre diferentes sistemas.

Figura 1

Modelo de intercambio de información a través de una API



Nota de "¿Qué significa REST?" por Appleute 2020 (<https://www.appleute.de/es/biblioteca-para-desarrolladores-de-aplicaciones/what-is-rest-api/>)

2.1.5.7 Docker

Según Merkel (2014), Docker es un proyecto de código abierto que permite empaquetar una aplicación junto con todas sus dependencias para ejecutarla de forma consistente en distintos entornos de desarrollo, prueba y producción.

Por otro lado, según Docker, Inc. Docker es un conjunto de productos que utiliza virtualización a nivel de sistema operativo para distribuir software empaquetado en contenedores, automatizando el despliegue de aplicaciones dentro de estos contenedores ligeros y permitiendo que se ejecuten de manera consistente en diferentes entornos de cómputo. El software central que ejecuta y administra estos contenedores se denomina Docker Engine

2.1.6 Herramientas de evaluación de rendimiento

2.1.6.1 JMeter

Según López y Rojas (2022) JMeter se utiliza principalmente para evaluar el comportamiento de aplicaciones web y servidores bajo condiciones de carga. Es altamente configurable, permitiendo definir distintos escenarios de prueba mediante árboles de pruebas y diversos elementos.

Así mismo Apache Software Foundation (2024) lo define como una herramienta de código abierto diseñada para realizar pruebas de carga funcional y de rendimiento en aplicaciones web. Puede simular múltiples usuarios simultáneos enviando solicitudes a un servidor y medir el rendimiento de la aplicación bajo diferentes condiciones de carga.

2.1.6.2 Grafana Lab

Según Chapman y Holmes (2023), Grafana es una herramienta central en los entornos modernos de observabilidad, ya que permite integrar diferentes fuentes de datos como Prometheus, InfluxDB o Elasticsearch en una única interfaz visual. Esta capacidad facilita el análisis de métricas del sistema, como el uso de CPU, memoria, tráfico de red o el comportamiento de aplicaciones distribuidas.

Por su parte, Elradi (2025) afirma que Grafana, al integrarse con herramientas como Prometheus y Node Exporter, proporciona una solución práctica y eficiente para monitorear entornos empresariales, especialmente cuando se requiere tomar decisiones rápidas basadas en métricas del sistema.

De forma similar, Hidalgo de Benito (2021) explica cómo esta herramienta puede adaptarse a diferentes entornos organizacionales, ofreciendo una alternativa flexible y accesible frente a otras soluciones propietarias.

2.2 Marco conceptual

2.2.1 Programación imperativa

Según Quintanar Perez, David (2024), la programación imperativa se centra en describir paso a paso cómo realizar una tarea, mediante una serie de instrucciones que modifican el estado del programa a lo largo del tiempo.

2.2.2 Programación reactiva

Kailash Nirmal (2025), define que la programación reactiva es un paradigma que permite desarrollar aplicaciones capaces de responder de forma rápida y eficiente a los cambios en los datos, mediante flujos de datos asíncronos y la propagación de eventos.

2.2.3 Persistencia de datos

Según Abraham Silberschatz, Peter Baer Galvin y Greg Gagne, la persistencia se manifiesta como la capacidad de los sistemas de archivos para guardar información en medios no volátiles (discos, SSDs). Los sistemas operativos se encargan de gestionar bloques de datos y metadatos para asegurar que los archivos y directorios permanezcan inalterados y accesibles hasta que se eliminan. Es la capa fundamental que habilita la persistencia a niveles más alto.

Por otro lado, Martin Fowler lo describe como un conjunto de componentes de software encargados de gestionar el almacenamiento y la recuperación de objetos de datos, abstraen los detalles del sistema de almacenamiento y permite que la lógica de negocio se concentre en sus tareas principales.

2.2.4 Tiempo de respuesta

Según Jader, M., Silva, R. (2019) define el tiempo de respuesta como el intervalo de tiempo que transcurre desde que se envía una solicitud de servicio hasta que la aplicación devuelve la respuesta completa, por ejemplo, el tiempo que tarda en cargarse una página web solicitada.

Además, LoadFocus (2013) aporta que es una medida del rendimiento de una transacción o consulta individual, define como el tiempo total que transcurre desde que el usuario envía la petición hasta que la aplicación indica que ha finalizado dicha petición.

2.2.5 Consumo de recursos

Según la norma ISO/IEC 25010, es considerado en que las cantidades y tipos de recursos (CPU, memoria, ancho de banda de red, etc.) que emplea un producto o sistema al ejecutar sus funciones cumpliendo las tareas definidas.

De la misma forma Tanenbaum, Andrew S., lo define como la utilización de los componentes físicos (CPU, memoria, dispositivos de entrada/salida) de una computadora por parte de los procesos de un programa. El sistema operativo juega un papel principal ya que es un gestor que asigna, protege y recupera estos recursos para que múltiples programas puedan compartirlos eficientemente sin interferir entre sí. En este contexto, el consumo de recursos se define como la demanda y el uso medible de la capacidad computacional y de almacenamiento disponible.

Sumado a ello, Silberschatz, Korth y Sudarshan (2020), explican que el consumo de recursos en bases de datos, como la demanda y utilización de los recursos computacionales (CPU, RAM, I/O de disco) del SGBD y sus componentes asociados (buffers, conexiones) para la ejecución de consultas, la gestión de transacciones y el mantenimiento de la persistencia de datos, que impactan la velocidad de respuesta y la concurrencia de las operaciones sobre los datos.

2.2.6 Latencia de operaciones

Andrew S. Tanenbaum, describe la latencia como el retraso asociado a las operaciones de entrada y salida (E/S) o al acceso a la memoria. La latencia de disco, por ejemplo, es el tiempo que tarda un disco duro en mover sus cabezales sobre el sector deseado (tiempo de búsqueda) y esperar a que el sector gire bajo el cabezal. En resumen, se refiere al tiempo que tarda la CPU en leer o escribir un dato en la memoria RAM. Una baja latencia en estas operaciones es crucial para el rendimiento general del sistema.

Por otro lado, Martin Kleppmann define que la latencia de las operaciones de lectura y escritura, al tiempo de una consulta a la base de datos es el tiempo que tarda el SGBD en procesar y devolver el resultado.

2.2.7 Throughput

Según James F. Kurose y Keith W. Ross, en el ámbito de las redes, definen el throughput como la tasa a la que los bits (o paquetes) que son transferidos exitosamente de un emisor a un receptor a través de una red o un enlace de comunicación. Para ellos, el throughput

efectivo es a menudo menor que el ancho de banda de un enlace debido a factores como la congestión de la red, los errores y los retrasos del protocolo.

Por otro lado, para Silberschatz, Korth y Sudarshan, el throughput de una base de datos es la cantidad de transacciones que el sistema de gestión de bases de datos (SGBD) puede procesar y confirmar exitosamente en un límite de tiempo. Esto incluye operaciones de lectura, escritura y actualización que garantizan las propiedades. Un alto TPS indica que la base de datos puede manejar una gran carga de trabajo concurrente.

2.2.8 SQL Server

Según Coronel y Morris (2018), SQL Server es un sistema de gestión de bases de datos relacional. Permite crear, administrar y consultar bases de datos mediante el uso de SQL.

2.2.9 Spring Web

Según Johnson, Hoeller y Arendsen (2022), Spring Web es un módulo del framework Spring que proporciona soporte para el desarrollo de aplicaciones web basado en el patrón MVC (Modelo-Vista-Controlador).

2.2.10 Spring WebFlux

Según Hoeller y Johnson (2022), Spring WebFlux es un módulo reactivo del framework Spring diseñado para manejar cargas concurrentes y asíncronas de manera eficiente, basado en flujos reactivos (Reactive Streams).

2.2.11 JMeter

Según Apache Software Foundation (2024), JMeter es una herramienta de código abierto diseñada para realizar pruebas de carga funcional y de rendimiento en aplicaciones web, capaz de simular múltiples usuarios simultáneos.

2.2.12 Grafana Lab

Según Chapman y Holmes (2023), Grafana es una herramienta que permite integrar diferentes fuentes de datos en una única interfaz visual, facilitando el análisis de métricas del sistema como el uso de CPU, memoria y tráfico de red.

Capítulo III

Materiales y métodos

3 Diseño metodológico

3.1 Tipo de investigación

La presente investigación es de tipo aplicada, debido a que busca resolver un problema concreto relacionado con el rendimiento en persistencia de datos de aplicaciones CRUD desarrolladas bajo dos paradigmas de programación: imperativo y reactivo, con el propósito de comparar las diferencias de rendimiento bajo las mismas condiciones de evaluación.

Según Bernal (2010), la investigación aplicada se orienta al estudio y solución de problemas reales mediante la utilización del conocimiento científico, buscando su aplicación práctica en contextos específicos. En ese sentido, esta investigación pretende aportar evidencia técnica útil para la toma de decisiones en el desarrollo de software.

3.2 Nivel de investigación

El nivel de investigación es descriptivo, ya que se orienta a medir, analizar y describir las características de rendimiento en persistencia de datos de aplicaciones CRUD desarrolladas bajo diferentes paradigmas de programación, considerando indicadores como tiempo de respuesta, throughput, latencia de operaciones y consumo de recursos.

Según Hernández, Fernández y Baptista (2014), los estudios descriptivos buscan especificar propiedades y características importantes de cualquier fenómeno que se analice. Esto se relaciona con el propósito de esta investigación, ya que se pretende medir indicadores de calidad como el tiempo de respuesta, el consumo de recursos, latencia y el throughput y describir las diferencias observadas entre ambos paradigmas bajo las mismas condiciones de evaluación.

3.3 Diseño de la investigación

La presente investigación adopta un diseño descriptivo-comparativo, debido a que se desarrollan dos aplicaciones CRUD funcionalmente equivalentes bajo distintos paradigmas de programación imperativo y reactivo, y posteriormente se comparan sus métricas de rendimiento bajo las mismas condiciones de evaluación.

Las aplicaciones implementadas fueron:

- a) Aplicación basada en el paradigma imperativo (Spring MVC + JPA)
- b) Aplicación basada en el paradigma reactivo (Spring WebFlux + R2DBC)

Ambas aplicaciones fueron evaluadas mediante pruebas de carga, estrés y picos utilizando Apache JMeter, manteniendo los mismos parámetros de prueba, recursos de infraestructura y condiciones de ejecución, con la finalidad de garantizar la objetividad y comparabilidad de los resultados obtenidos.

Vara Horna (2012) define los diseños descriptivos comparativos señalando que “tienen como objetivo lograr la identificación de diferencias o semejanzas con respecto a la aparición de un evento en dos o más grupos” y además “comparan características y variables entre dos o más estratos”. Asimismo, indica que “este diseño usa dos o más investigaciones descriptivas simples; esto es, recolectar información relevante en varias mediciones con respecto a un mismo fenómeno y luego comparar los datos recogidos”.

Se reconoce que el rendimiento en la persistencia de datos involucra múltiples componentes tecnológicos, tales como la infraestructura de red, el sistema operativo, el servidor de aplicaciones, los controladores de acceso a datos, la gestión de conexiones y el sistema gestor de base de datos. Por ello, a fin de que las diferencias observadas sean atribuibles al comportamiento de las aplicaciones evaluadas y no a variaciones del entorno, ambas aplicaciones fueron ejecutadas bajo condiciones idénticas: el mismo equipo local, el mismo sistema operativo, la misma instancia y configuración de SQL Server 2022, la misma base de datos y entidad, la misma asignación de recursos para los aplicativos, el mismo rango del pool de conexiones y los mismos escenarios y parámetros de prueba en Apache JMeter

3.4 Población y muestra

Por la naturaleza del diseño descriptivo-comparativo, la presente investigación no trabajó con una población ni con una muestra en sentido probabilístico, dado que no se pretende generalizar los resultados hacia un conjunto mayor de aplicaciones o ejecuciones. El análisis se efectuó de manera censal, sobre la totalidad de las mediciones de rendimiento registradas durante la evaluación de las dos aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.

En total se registraron 158 909 mediciones, distribuidas en 72 702 correspondientes a la aplicación imperativa y 86 207 a la aplicación reactiva, obtenidas en los escenarios de

pruebas de carga, estrés y picos ejecutados con Apache JMeter desde 1 hasta 20 usuarios virtuales durante 30 minutos por escenario, conforme al siguiente detalle:

Tabla 1

Tabla de cantidad de mediciones

Escenario	Aplicación imperativa	Aplicación reactiva
Prueba de carga	26 425	31 497
Prueba de estrés	19 878	24 672
Prueba de picos	26 399	30 038
Total	72 702	86 207

Nota. Elaboración propia.

Cada medición corresponde a la ejecución de una operación CRUD (Create, Read, Update, Delete), y sobre este conjunto completo de observaciones se calcularon las métricas de tiempo de respuesta, latencia, throughput y consumo de recursos (CPU y memoria JVM), por operación, escenario de prueba y aplicación evaluada.

3.5 Unidad de análisis

La unidad de análisis está constituida por cada medición de rendimiento registrada por operación CRUD, por escenario de prueba y por cada aplicativo CRUD desarrollado bajo paradigma de programación, imperativo y reactivo.

Cada unidad incluye valores de tiempo de respuesta, latencia y throughput obtenidos en una ejecución específica.

3.5.1 Aplicaciones desarrolladas

El paradigma de programación constituye el criterio de comparación del estudio y no una variable sujeta a medición: por diseño los dos casos evaluados, sin ser manipulado ni observado mediante instrumento alguno. Cada caso queda identificado por su implementación tecnológica, según el siguiente detalle:

Caso 1 - Aplicación con paradigma imperativo:

- a) Framework: Spring Boot 2.7.18 + Spring Web MVC
- b) Capa de persistencia: JPA
- c) Base de datos: Microsoft SQL Server 2022

- d) Modelo de ejecución: Síncrono y bloqueante

Caso 2 - Aplicación con paradigma reactivo:

- a) Framework: Spring Boot 2.7.18 + Spring WebFlux
- b) Capa de persistencia: R2DBC
- c) Base de datos: Microsoft SQL Server 2022
- d) Modelo de ejecución: Asíncrono y no bloqueante

Ambas aplicaciones se ejecutaron sobre Spring Boot 2.7.18 y Microsoft SQL Server 2022, implementando las mismas operaciones CRUD sobre una entidad idéntica, lo que garantiza su equivalencia funcional para efectos de la comparación.

3.6 Variable e indicadores

La investigación considera una única variable objeto de medición: el rendimiento en la persistencia de datos, evaluado en los dos casos descritos en la sección 3.5.1.

3.6.1 Definición conceptual de la variable

Variable de estudio

Rendimiento en la persistencia de datos: Corresponde al comportamiento observado durante la ejecución de operaciones CRUD sobre la base de datos y aplicativo, evaluado bajo las mismas condiciones de evaluación de carga, estrés y picos.

Dimensiones:

Para evaluar el rendimiento en la persistencia de datos se consideraron las siguientes métricas.

Dimensiones principales

- a. **Tiempo de respuesta:** Es el intervalo de tiempo que transcurre desde que se envía una solicitud de servicio hasta que la aplicación devuelve la respuesta.
- b. **Throughput:** Se define como la cantidad de transacciones que son transferidos exitosamente de un remitente a un receptor a través de una red o un enlace de comunicación.
- c. **Latencia de operaciones:** Se define el tiempo de operaciones de lectura y escritura, de una consulta a la base de datos en procesar y devolver el resultado hasta recibir el primer byte.

Dimensión complementaria

El consumo de recursos se considera como un indicador complementario de carácter descriptivo, utilizado para observar el comportamiento del sistema durante la ejecución de las pruebas.

- d. **Consumo de recursos:** Es considerado en que las cantidades y tipos de recursos (CPU, memoria, ancho de banda de red, etc.) que emplea un producto.

3.6.2 Definición operacional de la variable

Variable de estudio

Y: Rendimiento en persistencia de datos

Dimensiones:

- a. Tiempo de respuesta
- b. Throughput
- c. Latencia de operaciones
- d. Consumo de recursos (CPU y JVM)

Operacionalización de variable

Tabla 1

Matriz operacionalización de variable de estudio

Variable	Definición operacional	Dimensiones	Indicadores	unidad	instrumento
Rendimiento en la persistencia de datos	Comportamiento observado de cada aplicación CRUD durante la ejecución de operaciones sobre SQL Server,	Tiempo de respuesta	Tiempo transcurrido entre el envío de la solicitud y la recepción de la respuesta completa, por	Milisegundos	Apache Jmeter

medido bajo escenarios idénticos de carga, estrés y picos	operación CRUD			
Latencia de operaciones	Tiempo transcurrido hasta la recepción del primer byte de la respuesta, por operación CRUD y nivel de carga	Milisegundos	Apache Jmeter	
Throughput	Transacciones procesadas exitosamente e por unidad de tiempo	TPS	Apache Jmeter	
Consumo de recursos	Uso de CPU; consumo de memoria JVM (Heap)	% y MB	Grafana	

Nota. Elaboración propia.

3.7 Técnicas e instrumentos para el tratamiento de datos e información

3.7.1 Técnicas para recolectar información

La técnica principal utilizada fue la evaluación comparativa del rendimiento, mediante la ejecución de pruebas de carga, estrés y picos sobre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo, con el propósito de obtener y analizar métricas.

Como técnicas específicas se emplearon:

Pruebas de rendimiento: Son técnicas que miden la estabilidad, velocidad y capacidad de un sistema para mantener un nivel óptimo de funcionamiento bajo condiciones controlada. A través de estas pruebas se recopiló información sobre tiempos de respuesta, uso de recursos, throughput y latencia, permitiendo comparar el comportamiento de ambas aplicaciones bajo las mismas condiciones. En este estudio se aplicaron tres modalidades:

- a. Prueba de carga: Se evaluó el comportamiento del sistema bajo una cantidad constante de usuarios concurrentes durante un periodo sostenido de tiempo.
- b. Prueba de estrés: Se evaluó el comportamiento del sistema incrementando progresivamente la cantidad de usuarios concurrentes.
- c. Prueba de picos: Se evaluó el comportamiento del sistema ante incrementos repentinos y abruptos en la cantidad de usuarios concurrentes.

3.7.2 Instrumentos para recolectar información

Se utilizó Apache JMeter como instrumento principal para la generación de carga y la recolección de métricas de rendimiento.

Adicionalmente, se empleó Grafana para la visualización y monitoreo de métricas durante la ejecución de las pruebas.

3.7.3 Herramientas para el tratamiento de datos e información

Para el desarrollo y evaluación de las aplicaciones se utilizaron las siguientes herramientas, organizadas según su propósito:

Tabla 1

Herramientas para el tratamiento de datos

Software	Fabricante	Servicio
IntelliJ Idea	Jetbrains	Entorno de desarrollo integrado utilizado para escribir, depurar y ejecutar el código fuente de ambas aplicaciones.
Java	Oracle	Lenguaje de programación base sobre el que se

		desarrollaron ambas aplicaciones CRUD.
Spring	Spring, Inc.	Framework principal utilizado para construir ambas aplicaciones, proporcionando la configuración e infraestructura base.
Spring web	Spring, Inc.	Módulo utilizado para implementar la aplicación con paradigma imperativo, basado en el modelo síncrono y bloqueante.
Spring webflux	Spring, Inc.	Módulo utilizado para implementar la aplicación con paradigma reactivo, basado en el modelo asíncrono y no bloqueante.
Microsoft SQL Server 2022 Standar	Microsoft	Motor de base de datos relacional sobre el que ambas aplicaciones ejecutaron las operaciones CRUD evaluadas.
Docker	Docker, Inc.	Plataforma de software que permite empaquetar aplicaciones y las dependencias en unidades llamadas contenedores.

Nota. Elaboración propia.

Tabla 2

Herramientas de recolección y análisis de datos

Jmeter	Apache	Herramienta utilizada para simular los escenarios de prueba de carga, estrés y picos desde 1 hasta llegar a los 20 usuarios virtuales durante 30 minutos, registrando las métricas de tiempo de respuesta, throughput y latencia de cada operación CRUD.
--------	--------	--

Grafana Lab	Grafana	Herramienta utilizada para monitorear y visualizar en tiempo real el consumo de CPU y memoria JVM de cada aplicación durante la ejecución de las pruebas, integrando datos provenientes de Prometheus.
Prometheus	SoundCloud	Es un sistema de monitoreo que recopila y almacena métricas de aplicaciones, servidores, servicios para evaluar su rendimiento y generar alertas.
cAdvisor	Google	Es una herramienta que recopila información sobre el uso de contenedores como CPU, memoria, disco y red.
Sql_exporter	Código Libre	Es un exportador para Prometheus que obtiene métrica desde base de datos SQL.

Nota. Elaboración propia.

3.7.4 Procedimiento

Para evaluar el desempeño de los aplicativos CRUD desarrollados bajo los paradigmas imperativo y reactivo, se ejecutaron los siguientes pasos:

- Se desarrollaron dos aplicaciones CRUD con funcionalidades equivalentes, una basada en el paradigma imperativo utilizando Spring Web MVC y otra basada en el paradigma reactivo mediante Spring WebFlux, ambas conectadas a una misma base de datos SQL Server, cada aplicación CRUD como SQL server se desplegaron en docker para intentar mantener el ambiente controlado asignándoles recursos de RAM y CPU.
- Se definieron los escenarios de prueba, considerando carga sostenida, estrés incremental y picos de concurrencia. Para ello, se configuró la herramienta JMeter con 1 hasta llegar a los 20 usuarios virtuales y una duración de 30 minutos por cada prueba.

- c) Se ejecutaron las pruebas de rendimiento de manera independiente para cada aplicación, manteniendo las mismas condiciones de hardware y entorno de ejecución con el fin de garantizar la comparabilidad de los resultados.
- d) Se recopilaron las métricas de rendimiento, específicamente tiempo de respuesta, consumo de recurso, throughput y latencia mediante JMeter, mientras que el consumo de recursos del sistema, como CPU y memoria, fue monitoreado utilizando Grafana.
- e) Se realizó un análisis comparativo de los resultados obtenidos, considerando cada tipo de prueba y cada operación CRUD implementada en ambas aplicaciones.
- f) Finalmente, se formularon conclusiones con base en el análisis de los resultados, describiendo y comparando las diferencias observadas en el rendimiento de las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en los diferentes escenarios evaluados.

3.7.5 Técnicas para el procesamiento y análisis de datos

En esta sección describe cómo se transformaron los datos crudos obtenidos de JMeter y Grafana en información comparable y analizable.

3.7.5.1 Procesamiento de datos

Las métricas de rendimiento se organizaron por operación CRUD, tomando como fuente los resultados del reporte resumen de JMeter.

Asimismo, los datos de consumo de CPU y memoria JVM fueron extraídos de los paneles de monitoreo de Grafana y posteriormente tabulados por escenario de prueba.

Finalmente, los valores recopilados se consolidaron en tablas comparativas, agrupando los resultados correspondientes a la aplicación CRUD construida bajo el paradigma imperativa frente a la construida bajo el paradigma reactiva, con el fin de facilitar el análisis del rendimiento.

3.7.5.2 Análisis de datos

Se calcularon valores estadísticos descriptivos básicos para cada métrica evaluada:

- a) Valor mínimo
- b) Valor máximo
- c) Promedio

- d) Desviación estándar
- e) Mediana
- f) Percentil 90
- g) Percentil 95

Posteriormente, se elaboraron tablas comparativas por indicador (tiempo de respuesta, throughput, latencia, consumo de CPU y memoria JVM) para cada tipo de prueba ejecutada.

Finalmente, los resultados fueron interpretados mediante análisis comparativo técnico, identificando las diferencias de rendimiento observadas en cada indicador y escenario de prueba.

Capítulo IV

Resultados y discusión

4 Análisis y resultados de la investigación

En este capítulo se presentan los resultados obtenidos a partir de las pruebas de rendimiento realizadas sobre las aplicaciones desarrolladas bajo los paradigmas imperativo y reactivo.

Los datos mostrados corresponden a los valores procesados obtenidos de las ejecuciones de las pruebas de rendimientos descritas en el capítulo anterior, organizados por indicador de rendimiento, tipo de prueba y operación CRUD.

4.1 Arquitectura de las aplicaciones CRUD construidas

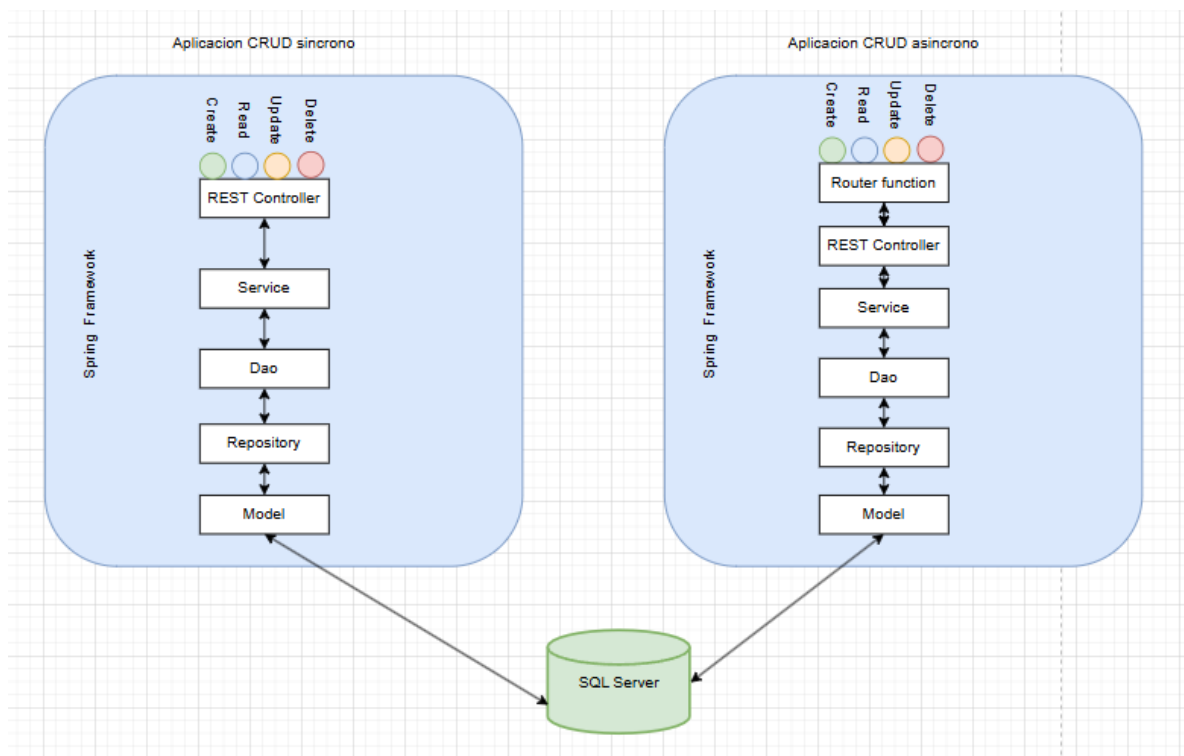
Para poder realizar las operaciones CRUD se realizó dos aplicaciones:

- Aplicación Imperativa: Se realizó de manera sincrónica y de manera secuencial.
- Aplicación Reactiva: Se realizó de manera asincrónica y no bloqueantes.

A continuación, se muestra la arquitectura de las 2 aplicaciones CRUD construidas:

Figura 2

Arquitectura de las aplicaciones CRUD



Ambas contienen una misma arquitectura como se muestra en la figura 2, a continuación, se procede a explicar la estructuración:

- a. **Rest Controller:** Capa encargada de recibir las peticiones HTTP (GET, POST, PUT, DELETE), es la primera entrada a la aplicación.
- b. **Service:** Capa que contiene la lógica del negocio, acá se aplica las reglas, lógica, validación.
- c. **Dao:** Esta capa contiene la lógica que se realizara con los datos extraído convirtiendo a objetos más sencillos para trabajar en otras capas.
- d. **Repository:** Esta capa interactúa directamente con la base de datos.
- e. **Model:** Representa entidades, donde se mapea tablas de la base de datos.

Sin embargo, una única diferencia es que en la reactiva tiene una capa extra que es el Router Function esta sirve para definir rutas en aplicaciones reactivas con webflux.

4.2 Código fuente de las aplicaciones

A continuación, se muestra el código fuente de la aplicación construida de manera sincrónica (paradigma imperativo):

Controller:

```

@RestController
@RequestMapping("/api/bs")
public class CuentaController {

    @Autowired
    private CuentaService service;

    @GetMapping
    public List<Cuenta> getAll() {
        return service.getAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> getById(@PathVariable Long id) {
        Cuenta cuenta= service.obtenerCuentaRelacionada(id);
        return ResponseEntity.status(HttpStatus.OK).body(cuenta);
    }

    @PostMapping
    public ResponseEntity<Cuenta> create(@RequestBody
CuentaRelacionadaDTO cuenta) {
        return ResponseEntity.status(HttpStatus.CREATED)
            .body(service.crearCuentaRelacionada(cuenta));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Cuenta> update(@PathVariable Long id,
                                           @RequestBody
CuentaRelacionadaDTO cuentaRelacionadaDTO) {
        return ResponseEntity.ok(service.actualizarCuenta(id,
cuentaRelacionadaDTO));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
        service.eliminarCuentaRelacionada(id);
        return ResponseEntity.noContent().build();
    }
}

```

A continuación, se muestra el código fuente de la aplicación construida de manera asincrónica (paradigma reactivo):

MasterHandler:

```

@Component
public class MasterHandler {

    @Autowired
    private CuentaService cuentaService;

    public Mono<ServerResponse> crear(ServerRequest request) {
        Mono<CuentaRelacionadaDTO> cuentaRelacionada =
            request.bodyToMono(CuentaRelacionadaDTO.class);

        return cuentaRelacionada.flatMap(dto ->
            cuentaService.crearCuentaRelacionada(dto)
                .flatMap(cuentaCreada -> ServerResponse.ok()
                    .contentType(MediaType.APPLICATION_JSON_UTF8)
                    .body(fromObject(cuentaCreada))));
    }

    public Mono<ServerResponse> obtener(ServerRequest request){
        Long id = Long.valueOf(request.pathVariable("id"));
        return cuentaService.obtenerCuentaRelacionada(id)
            .flatMap(c -> ServerResponse
                .ok()
                .contentType(MediaType.APPLICATION_JSON_UTF8)
                .body(fromObject(c)))
            .switchIfEmpty(ServerResponse.notFound().build());
    }

    public Mono<ServerResponse> editar(ServerRequest request){
        Mono<CuentaRelacionadaDTO> cuentaRelacionada =
request.bodyToMono(CuentaRelacionadaDTO.class);
        Long id = Long.valueOf(request.pathVariable("id"));
        return cuentaRelacionada.flatMap(dto -> {
            return cuentaService.actualizarCuenta(id,dto)
                .flatMap(cuentaEditada -> ServerResponse.ok()

.contentType(MediaType.APPLICATION_JSON_UTF8)
                    .body(fromObject(cuentaEditada)))

.switchIfEmpty(ServerResponse.notFound().build());
        });
    }

    public Mono<ServerResponse> eliminar(ServerRequest request){
        Long id = Long.valueOf(request.pathVariable("id"));
        Mono<Cuenta> cuenta = cuentaService.getId(id);
        return cuenta.flatMap(c->
cuentaService.eliminarCuentaRelacionada(c.getIdCuenta()).then(Server
Response.noContent().build()))
            .switchIfEmpty(ServerResponse.notFound().build());
    }
}

```

4.3 Detalles de los endpoints

A continuación, se da más detalles de los endpoints implementados.

Tabla 3

Detalle del servicio POST para la creación de data.

Nombre	Crear
URL	http://{domain}:{port}/api/bs
Método HTTP	POST
Parámetros	<pre>{ "cliente": { "idCliente": <Integer> }, "cuenta": { "tipoCuenta": <String>, "numeroCuenta": <String>, "estado": <Integer>, "fechaApertura": <Datetime> }, "saldo": { "saldoActual": <BigDecimal> }, "transacciones": [{ "idTipo": <Integer>, "monto": <BigDecimal>, "descripcion": <String> }, { "idTipo": <Integer>, "monto": <BigDecimal>, "descripcion": <String> }] }</pre>

Descripción	Este endpoint es el encargado de crear la información de la cuenta, saldo, y movimientos.
--------------------	---

Nota. Elaboración propia

Tabla 4

Detalle del servicio GET para la obtención de data.

Nombre	Obtener
URL	http://{domain}:{port}/api/bs/{id}
Método HTTP	GET
Parámetros	[Ninguno]
Descripción	Este endpoint es el encargado de obtener la información de la cuenta con su respectivos saldo, cliente y movimientos.

Nota. Elaboración propia

Tabla 5

Detalle del servicio PUT para la edición de data.

Nombre	Actualización
URL	http://{domain}:{port}/api/bs/{id}
Método HTTP	PUT
Parámetros	{ "cliente": { "idCliente": <Integer> }, "cuenta": { "tipoCuenta": <String>, "numeroCuenta": <String>, "estado": <Integer>, "fechaApertura": <Datetime> }, "saldo": { "saldoActual": <BigDecimal> }

	<pre> }, "transacciones": [{ "idTipo": <Integer>, "monto": <BigDecimal>, "descripcion": <String> }, { "idTipo": <Integer>, "monto": <BigDecimal>, "descripcion": <String> }] } </pre>
--	---

Descripción	Este endpoint es el encargado de la actualización de la información de la cuenta como saldo y movimientos.
--------------------	--

Nota. Elaboración propia

Tabla 6

Detalle del servicio DELETE para la eliminación de data.

Nombre	Eliminación
URL	http://{domain}:{port}/api/bs/{id}
Método HTTP	DELETE
Parámetros	[Ninguno]
Descripción	Este endpoint es el encargado de eliminar la información de la cuenta con su respectivos saldo y movimientos.

Nota. Elaboración propia

4.4 Tablas de la base de datos

Para este desarrollo se utilizó una base de datos relacional (SQL Server), ambas aplicaciones se están conectando a una misma base de datos, está conformado por 4 tablas de las cuales se trabajó con 3 tablas y son las siguientes tablas:

- a. Cuentas: Tabla encargada de almacenar información de las cuentas.
- b. Saldos: Tabla encargada de almacenar información de los saldos.
- c. Transacciones: Tabla encargada de almacenar información de los movimientos.

A continuación, se muestra el DDL de las tablas:

- a. Para la tabla Cuentas

```
CREATE TABLE [dbo].[cuentas](
    [id_cuenta] [int] IDENTITY(1,1) NOT NULL,
    [id_cliente] [int] NULL,
    [tipo_cuenta] [nvarchar](20) NULL,
    [numero_cuenta] [nvarchar](20) NULL,
    [fecha_apertura] [datetime] NULL,
    [estado] [nvarchar](20) NULL,
    CONSTRAINT [PK__cuentas__C7E28685469FA3FA] PRIMARY KEY CLUSTERED
(
    [id_cuenta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[cuentas] WITH CHECK ADD CONSTRAINT
[FK__cuentas__id_clie__276EDEB3] FOREIGN KEY([id_cliente])
REFERENCES [dbo].[clientes] ([id_cliente])
GO
```

- b. Para la tabla Saldos

```
CREATE TABLE [dbo].[saldos](
    [id_saldo] [int] IDENTITY(1,1) NOT NULL,
    [id_cuenta] [int] NULL,
    [saldo_actual] [decimal](15, 2) NULL,
    [ultima_actualizacion] [datetime] NULL,
    PRIMARY KEY CLUSTERED
(
    [id_saldo] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[saldos] WITH CHECK ADD CONSTRAINT
[FK__saldos__id_cuent__2C3393D0] FOREIGN KEY([id_cuenta])
REFERENCES [dbo].[cuentas] ([id_cuenta])
```

c. Para la tabla Transacciones

```
CREATE TABLE [dbo].[transacciones](
    [id_transaccion] [int] IDENTITY(1,1) NOT NULL,
    [id_cuenta] [int] NULL,
    [id_tipo] [int] NULL,
    [monto] [decimal](15, 2) NULL,
    [fecha] [datetime] NULL,
    [descripcion] [nvarchar](255) NULL,
    PRIMARY KEY CLUSTERED
    (
        [id_transaccion] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[transacciones] WITH CHECK ADD CONSTRAINT
[FK__transacci__id_cu__31EC6D26] FOREIGN KEY([id_cuenta])
REFERENCES [dbo].[cuentas] ([id_cuenta])
GO
```

4.5 Presentación de los resultados

Para obtener los resultados se realizaron 3 pruebas de performance los cuales son:

- a. Carga: Se realizó para medir el comportamiento del sistema con usuarios concurrente constante.
- b. Estrés: Se realizó para medir el comportamiento del sistema con usuarios concurrentes e incrementando constante.
- c. Picos: Se realizó para medir el comportamiento del sistema con incrementos repentinos de usuarios.

Estas pruebas fueron realizadas para ambos sistemas contruidos, para poder evaluar el tiempo de respuesta, consumo de recursos, latencia y el throughput, para obtener los resultados se realizaron con la herramienta JMeter y Grafana Lab.

Posterior a la ejecución de las pruebas, se realizó el análisis de los resultados obtenidos a partir de las métricas recolectadas mediante las pruebas, este análisis permitió comparar el comportamiento de ambos sistemas. Con base en estas evidencias, se formularán conclusiones a partir del análisis comparativo del rendimiento de los aplicativos desarrollados bajo los paradigmas de programación imperativo y reactivo, identificando las diferencias observadas en cada uno de los escenarios de concurrencia evaluados.

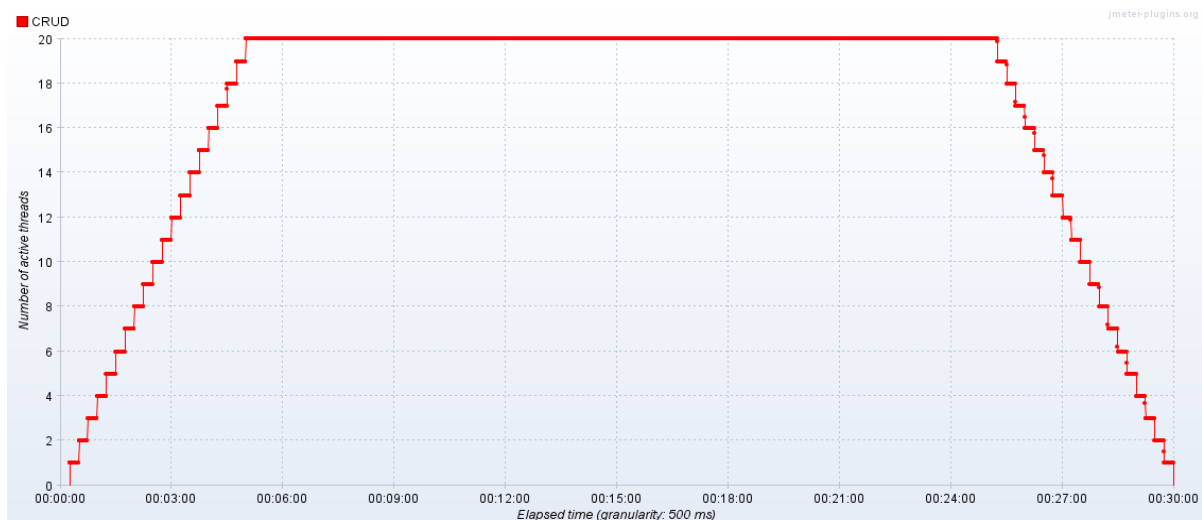
Para mayor claridad, desde ahora se llamará sincrónica al aplicativo construido con el paradigma imperativo y asíncrono lo llamaremos al aplicativo construido con el paradigma reactivo.

4.5.1 Pruebas de carga

Según se ilustra en la figura 3, esta prueba de carga se ejecutaron 20 hilos (usuarios virtuales) con una duración de 30 minutos, de los cuales se comenzó con 0 usuario y aumentándose gradualmente en los primeros 5 minutos hasta llegar a los 20 usuarios, teniendo así una totalidad de 20 usuarios por 20 minutos, y después en los últimos 5 minutos disminuyendo gradualmente hasta llegar al 0 usuario.

Figura 3

Prueba de carga



4.5.1.1 Resultados del aplicativo sincrónico

Los resultados que se obtuvieron para el aplicativo sincrónico fueron los siguientes:

Se detalla a continuación sobre cada endpoint sus métricas:

Tabla 7

Resultados del aplicativo sincrónico en la prueba de carga

Endpoint	Cantidad de ejecuciones	Tiempo de respuesta promedio (ms)	Tiempo mínimo de respuesta (ms)	Tiempo máximo de respuesta (ms)	Desviación estándar	Throughput	Mediana	90%.	95%
Creación	6612	1288	321	3199	359.61	3.7/sec	1242	1770	1928
Obtener	6608	405	5	2056	261.11	3.7/sec	377	737	862
Edición	6607	1787	390	3969	473.59	3.7/sec	1752	2404	2610
Eliminación	6598	1054	231	2659	335.00	3.7/sec	1007	1509	1653
Total	26425	1134	5	3969	616.90	14.8/sec	1112	1967	2237

Nota. Elaboración propia.

Se detalla a continuación sobre cada endpoint sus métricas:

- a) **Resultado del endpoint Creación:** Se ejecutó 6612 veces, con un promedio de respuesta de 1288 ms, el mínimo tiempo de respuesta fue de 321 ms y el máximo de 3199 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 359.61, con un throughput de 3.7/sec., con una mediana de 1242 y percentil 90 de 1770 y percentil 95 de 1928 en JMeter.
- b) **Resultado del endpoint Obtener:** Se ejecutó 6608 veces, con un promedio de respuesta de 405 ms, el mínimo tiempo de respuesta fue de 5 ms y el máximo de 2056 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 261.11, con un throughput de 3.7/sec., con una mediana de 377 y percentil 90 de 737 y percentil 95 de 862 en JMeter.

- c) **Resultado del endpoint Edición:** Se ejecutó 6607 veces, con un promedio de respuesta de 1787 ms, el mínimo tiempo de respuesta fue de 390 ms y el máximo de 3969 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 473.59, con un throughput de 3.7/sec., con una mediana de 1752 y percentil 90 de 2404 y percentil 95 de 2610 en JMeter.
- d) **Resultado del endpoint Eliminación:** Se ejecutó 6598 veces, con un promedio de respuesta de 1054 ms, el mínimo tiempo de respuesta fue de 231 ms y el máximo de 2659 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 335.00, con un throughput de 3.7/sec., con una mediana de 1007 y percentil 90 de 1509 y percentil 95 de 1653 en JMeter.

4.5.1.1.1 Latencia

a) Latencia durante la prueba

En la siguiente figura 4 se muestra la latencia promedio en el tiempo transcurrido de la prueba para las 4 operaciones CRUD. El gráfico muestra latencia promedio vs el tiempo de la prueba, en la cual mide específicamente el tiempo hasta recibir el primer byte de respuesta del servidor.

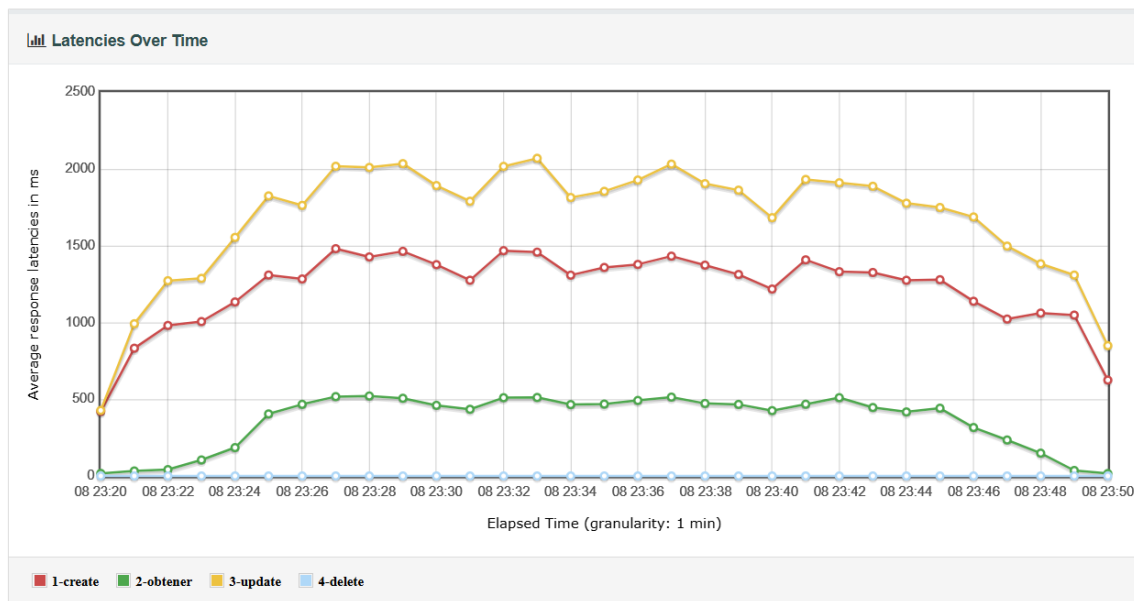
Se observa un comportamiento similar para cada operación donde hay una fase de ascenso esto sería normal ya que en los primeros minutos de la prueba específicamente 5 minutos va creciendo los usuarios hasta llegar a los 20 usuarios, seguida de una fase de estable de 20 minutos y un descenso en los últimos 5 minutos ya que los usuarios están disminuyendo para terminar la prueba.

Por operación se detalla lo siguiente:

- a) **Create (rojo):** Se ubica en un rango intermedio, estabilizándose entre 1200 a 1400ms con un pico máximo de 1481 en el minuto 23:27.
- b) **Read (verde):** Se muestra una latencia considerablemente menor con respecto a los endpoint de create y update, estabilizándose entre 400 a 500ms, esto normal ya que las operaciones de lectura son más rápidas.
- c) **Update (amarillo):** Presenta una latencia más alta de las 4, estabilizándose entre 1500 - 1900ms, con un pico de máximo de 2070 en el minuto 23:30.
- d) **Delete (azul):** Se mantiene en 0ms durante la prueba ya que para esta operación no se espera ni una respuesta(response) que tenga contenido, es por eso que la respuesta no tiene un byte para poder calcularlo.

Figura 4

Gráfica de latencia del aplicativo sincrónico



4.5.1.1.2 Response Time

a) Tiempo de respuesta durante la prueba

En el siguiente gráfico 5 se muestra el tiempo de respuesta promedio en el tiempo que duro la prueba, se puede observar el grafico de las 4 operaciones (CRUD). El tiempo de respuesta mide el proceso completo de la solicitud, desde el envío hasta la recepción total de la respuesta.

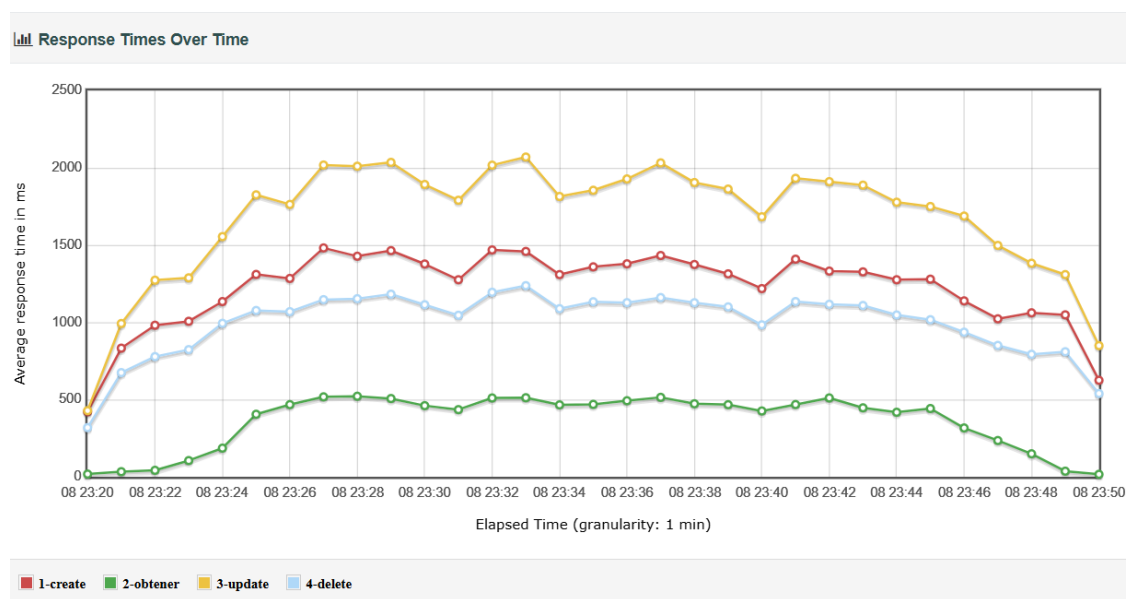
En el gráfico se identifican también 3 fases que la primera es la de creciente o ascenso en la cual se puede observar que en los primeros minutos hasta el minuto 5 se muestra una elevación ya que en esos momentos se están incrementando los usuarios hasta llegar hasta los 20 usuarios, después se mantiene una estabilidad durante los 20 minutos ya que se encuentra activos los 20 usuarios, después una fase de descenso en la cual los usuarios van decreciendo para terminar la prueba. Se detalla por operación:

- a) **Create (rojo):** Se ubica en un rango de 1000 a 1450, con el punto más alto en 1490 en el tiempo 23:27. Estos comportamientos es propio de una operación de escritura que requiere insertar registros a las tablas de cuenta, saldo y operación en todo el proceso.

- b) **Read (verde):** Es la operación más rápida, ya que oscila entre 150 a 510 durante toda la fase. Esto es rápido ya que no bloquea las tablas al momento de consultar. Teniendo un punto máximo de 521ms en el tiempo 23:28.
- c) **Update (amarillo):** Presentan el tiempo de respuesta más alto, estabilizándose entre 1700 y 2070ms, con su pico máximo de 2070ms en el tiempo 23:33 de la prueba, esto debido que busca los registros en cada tabla, realiza la modificación y guardado del registro en las tablas teniendo una mayor sobrecarga transaccional.
- d) **Delete(azul):** Es el proceso que se ubica en una estabilización de 700 a 1150, un valor estable ya que se encuentra en tiempos menores de create y update ya que este proceso se encarga de buscar y eliminar el registro, teniendo un tiempo de respuesta más alto en 1237 en el tiempo 23:33.

Figura 5

Gráfica de tiempo de respuesta durante la prueba



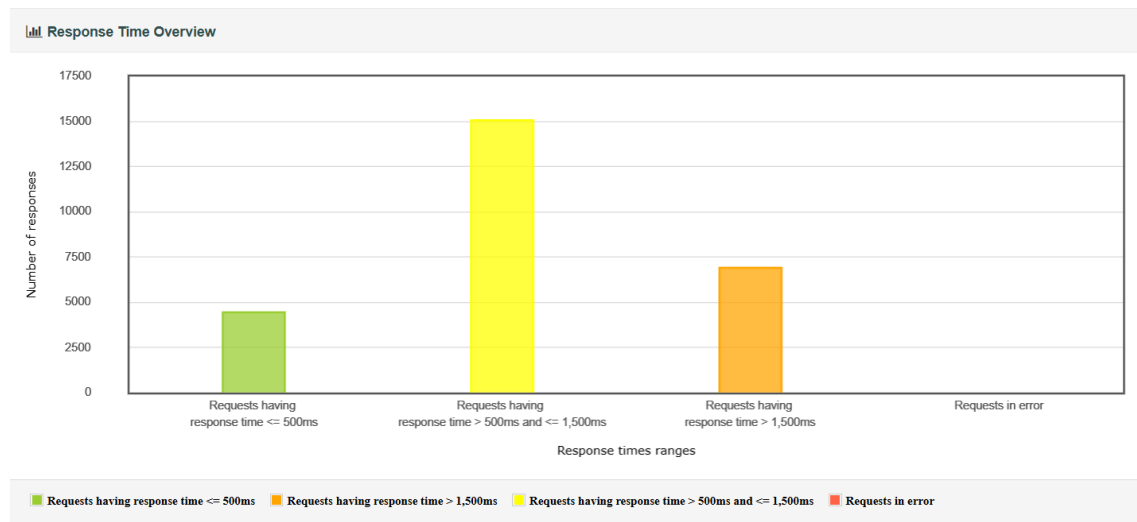
b) Distribución de cantidad de peticiones y tiempo de respuesta

En la siguiente figura 6 se muestra la distribución general de cantidades de respuestas, agrupando según el rango de tiempo que duro la peticiones se dividen en 3 partes:

- a) Tiempo de respuesta de ≤ 500 ms: Se 4440 peticiones
- b) Tiempo de respuesta entre $500 < > 1500$: 15080 peticiones
- c) Tiempo de respuesta mayor de >1500 : 6905 peticiones

Figura 6

Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico



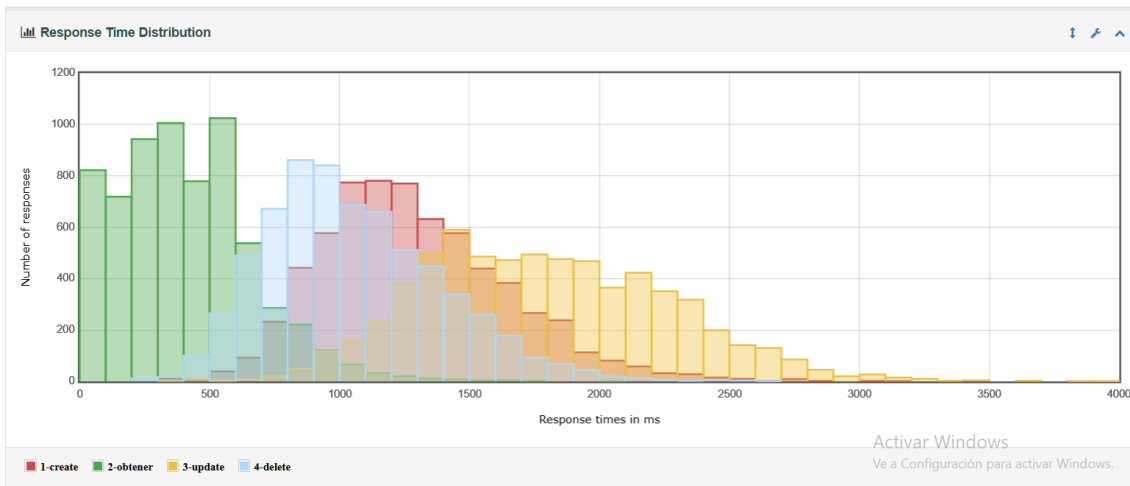
c) Distribución de tiempo de respuesta

En la siguiente figura 7 se muestra la distribución de frecuencias de tiempo de respuesta, separada por operación CRUD.

- Create (rojo):** Se distribuye en mayo concentración en 900 a 1300ms, teniendo la mayor distribución en el rango de 781 en 1100 a 1200ms.
- Read (verde):** Varias peticiones se concentran en el rango de 0 a 600ms, teniendo en la mayor distribución de 1025 peticiones en 500 a 600 ms.
- Update (amarillo):** Se distribuye en mayor concentración en 1300 a 1900ms, teniendo la mayor distribución con 590 peticiones en 1400 a 1500ms.
- Delete(azul):** Se distribuye principalmente en 700 a 1300ms, teniendo en la mayoría 862 peticiones en 800 a 900ms.

Figura 7

Gráfica de distribución de tiempo de respuesta



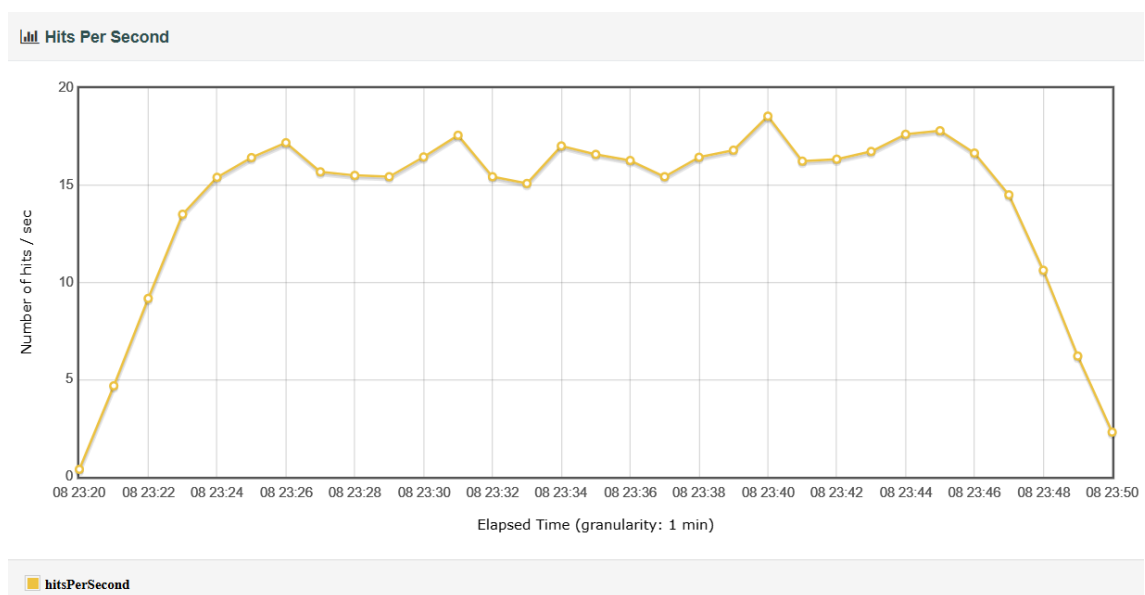
4.5.1.1.3 Throughput

a) TPS en general

En la siguiente figura 8 se puede observar el total de los TPS en el transcurso de la prueba como va variando en todo el transcurso, en este caso es la suma general de las 4 operaciones CRUD, se puede observar que hay un ascenso cuando empieza en los primeros minutos es normal ya que varios usuarios están siendo activados para realizar peticiones y dentro de los 20 minutos se mantiene estable manteniendo un rango de 15 a 17 TPS y teniendo un pico máximo de 18.57 en el tiempo 23:40 y en los últimos 5 minutos teniendo un descenso ya que usuarios se van disminuyendo.

Figura 8

Gráfica de TPS en general del aplicativo sincrónico



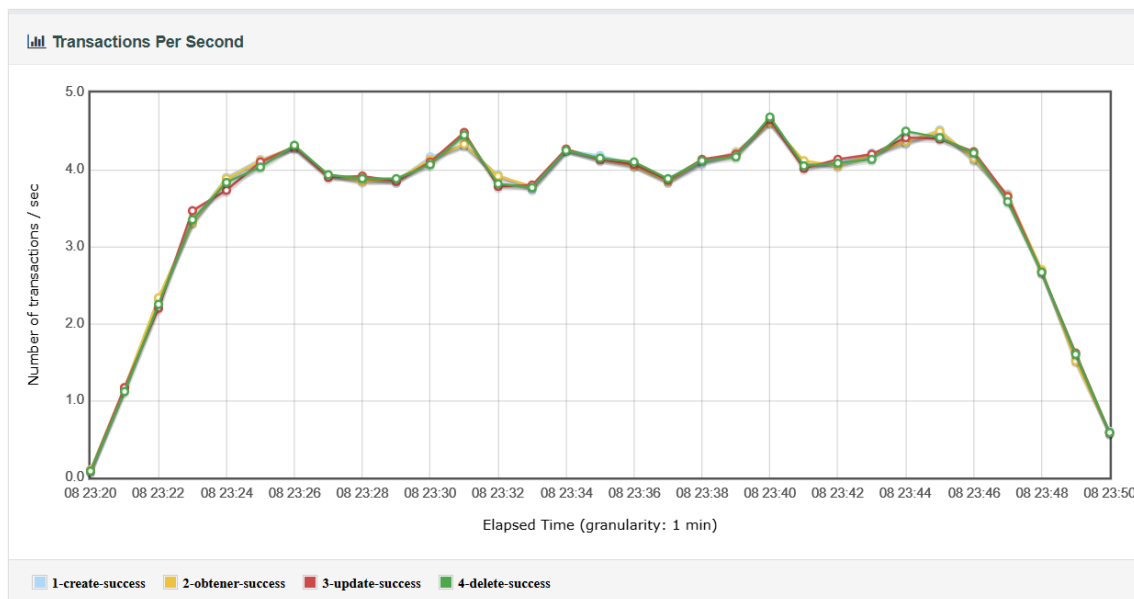
b) TPS por operación

En la siguiente figura 9 se muestra el throughput (transacciones por segundo) de las cuatro operaciones CRUD a lo largo del tiempo de prueba. Aquí se observa que las 4 operaciones se comportan de manera prácticamente idénticas.

- a) **Create (azul):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 23:4.
- b) **Read (amarillo):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 23:4.
- c) **Update (rojo):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 23:4.
- d) **Delete(verde):** Se puede observar que alcanza su punto máximo de 4.7 en el tiempo de 23:4.

Figura 9

Gráfica del TPS por operación del aplicativo sincrónico



4.5.1.1.4 Consumo de recursos

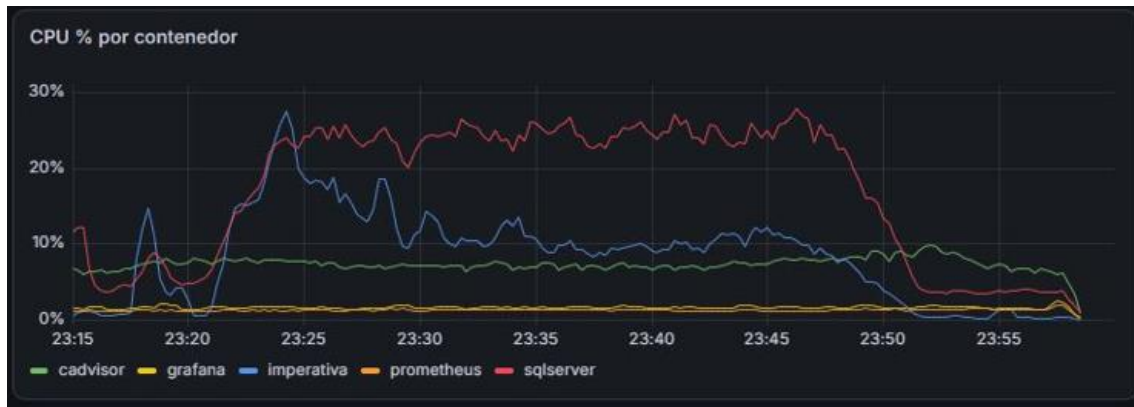
a) Consumo de CPU

En la siguiente figura 10 se observa el consumo de CPU por contenedor Docker durante los 30 minutos de prueba. El contenedor de SQL Server (rojo) presenta el consumo más alto, alcanzando picos de aproximadamente 27-28% durante la fase estable (23:25-23:45), mientras que el contenedor de la aplicación imperativa (azul) se mantiene en un rango de

8-15%, con un pico de 19% al inicio de la prueba (23:20). Los contenedores de monitoreo (cAdvisor, Grafana, Prometheus) presentan un consumo pequeño, por debajo del 5%, confirmando que no interfieren en la medición.

Figura 10

Gráfica del consumo de CPU del aplicativo sincrónico

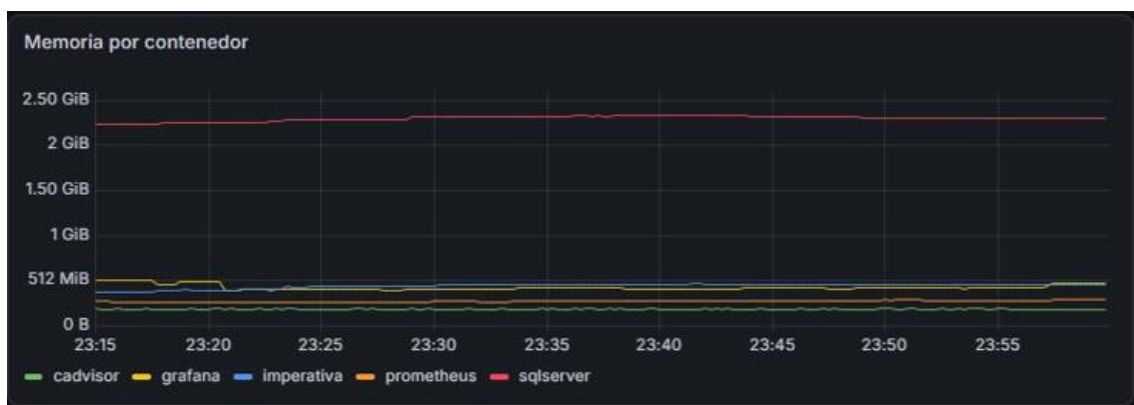


b) Consumo de memoria

En la siguiente figura 11 se muestra la memoria se mantiene prácticamente constante durante toda la prueba, sin picos ni variaciones asociadas al aumento de carga. El contenedor de SQL Server (rojo) consume de forma estable 2.5 GiB, mientras que la aplicación imperativa (azul) se mantiene alrededor de 400-512 MiB.

Figura 11

Gráfica de consumo de memoria del aplicativo sincrónico



c) Consumo de JVM

1. JVM Heap Memory

En la siguiente figura 12 se muestra el uso del heap de la JVM de manera desagregado por generación: G1 Eden Space, G1 Old Gen y G1 Survivor Space,

junto con el límite máximo configurado. Los tres componentes se mantienen en valores estables durante toda la prueba, lo que indica que la aplicación está operando normal sin alcanzar su capacidad máxima de heap asignada. Esto significa que, para esta carga de prueba con 20 usuarios concurrentes, la memoria no constituye un factor que limite el rendimiento observado, y que existe un amplio margen antes de que la gestión de memoria se convierta en un cuello de botella.

Figura 12

Gráfica de consumo de JVM Heap Memory del aplicativo sincrónico

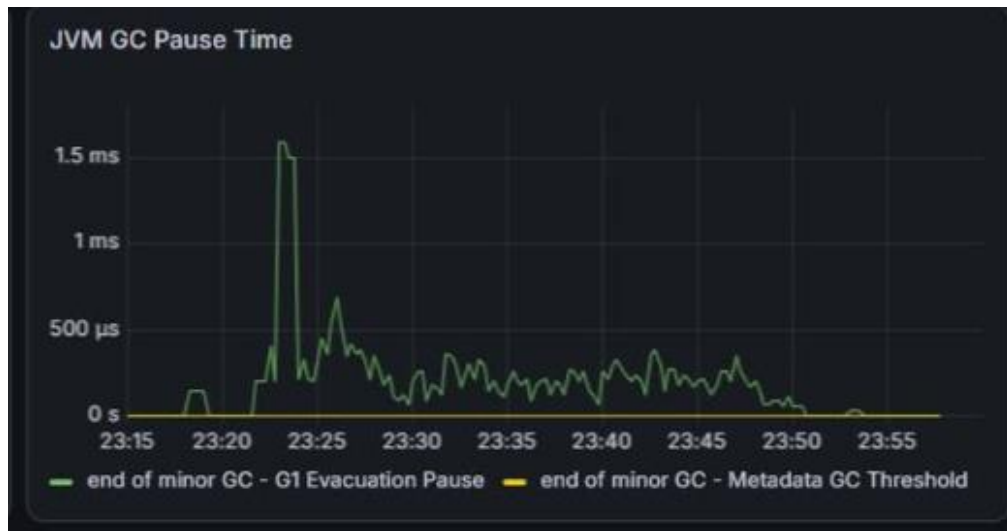


2. JVM GC Pause Time

En la siguiente figura 13 se observa el tiempo de pausa generado por el recolector de basura (Garbage Collector) durante la ejecución. Las pausas se mantienen mayormente por debajo de 500 microsegundos a lo largo de la prueba, con un pico de aproximadamente 1.5 ms alrededor del minuto 23:25, coincidiendo con el inicio de la prueba. El comportamiento del GC es regular y de bajo impacto, lo que indica que la gestión de memoria de la JVM no representa una fuente significativa de latencia adicional para las operaciones CRUD evaluadas.

Figura 13

Gráfica de consumo de JVM Pause Time del aplicativo sincrónico



3. JVM Threads

En la siguiente figura 14 se muestra el número de hilos vivos (verde) e hilos daemon (amarillo) de la JVM. Se observa un incremento desde 20 hilos en reposo hasta estabilizarse entre 35-38 hilos activos durante la fase de carga en el tiempo (23:25-23:45), y luego un descenso a los valores base al finalizar la prueba. Este comportamiento es el esperado, donde la petición HTTP entrante ocupa un hilo dedicado durante toda su ejecución de procesamiento + espera de respuesta de la base de datos. El número de hilos supera ese valor ya que tiene valor de 35-38 esto refleja un adicional de hilos internos del framework y del contenedor de aplicaciones Tomcat y no solo los hilos dedicados directamente a las peticiones de usuario.

Figura 14

Gráfica de consumo de JVM Threads del aplicativo sincrónico



d) Conexiones HikariCP

En la siguiente figura 15 se muestra el estado del pool de conexiones a la base de datos, conexiones activas (verde), idle/libres (amarillo) y en espera/pending (azul). Se observa que, al iniciar la carga en el tiempo 23:25, las conexiones activas alcanzan rápidamente su máximo configurado de 15 conexiones y se mantienen en ese tope durante toda la fase, mientras que simultáneamente aparecen conexiones en espera (pending), estabilizándose en aproximadamente 5. Esto indica que el pool de conexiones se saturó bajo la carga de 20 usuarios concurrentes, y que existían peticiones adicionales esperando a que se liberó una conexión antes de poder ejecutar la siguiente operación contra la base de datos generando un cuello de botella directamente y esto es atribuible al tamaño configurado del pool y al modelo de conexión bloqueante, donde cada hilo retiene su conexión durante todo el ciclo de vida de la petición.

Figura 15

Gráfica de conexiones de HikariCP del aplicativo sincrónico



e) Tiempo de espera de conexión

En la siguiente figura 16 se muestra el tiempo que una petición debe esperar para obtener una conexión libre del pool. Se observa un salto de 0 ms a aproximadamente 5 segundos constantes entre los minutos 23:25 y 23:45. Esto significa que, durante gran parte de la prueba, una porción significativa del tiempo total de respuesta de la aplicación no correspondía al procesamiento real de la operación CRUD, sino a la espera por obtener una conexión disponible hacia SQL Server.

Figura 16

Gráfica de tiempo de espera de conexión del aplicativo sincrónico



f) Throughput SQL Server

En la siguiente figura 17 se muestran dos métricas batch requests/sec (verde), que mide lotes de comandos SQL ejecutados, y transacciones/sec (amarillo). Ambas curvas muestran un crecimiento sostenido desde el inicio de la prueba hasta estabilizarse alrededor del minuto 23:50, alcanzando aproximadamente 450K batch requests acumulados y 100K transacciones acumuladas.

Figura 17

Gráfica de Throughput del consumo de sql server del aplicativo sincrónico



g) Locks y deadlocks

En la siguiente figura 18 se observa un crecimiento sostenido de los lock waits durante la fase de carga, alcanzando aproximadamente 13.5K esperas acumuladas antes de estabilizarse, mientras que la línea de deadlocks se mantiene prácticamente en 0 durante toda la prueba. Esto indica que, si existieron esperas por bloqueos de fila/tabla y no se registraron deadlocks que hubieran obligado a SQL Server a abortar transacciones, lo cual es confiable con la tasa de error del 0% observada en JMeter.

Figura 18

Gráfica de lock y deadlock del aplicativo sincrónico



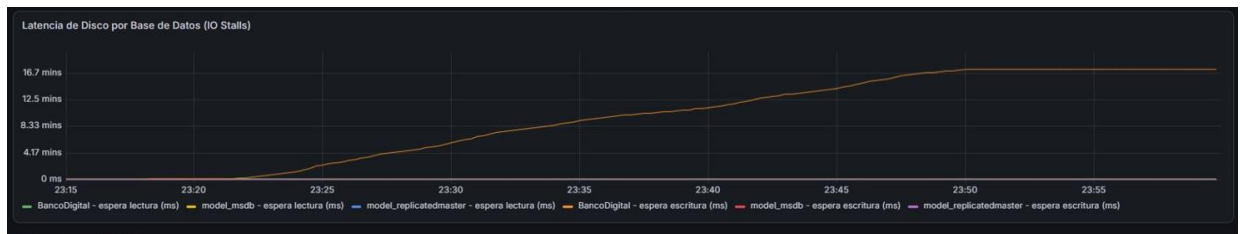
h) Latencia de Disco por base de datos

En la siguiente figura 19 se mide el tiempo acumulado de espera en operaciones de lectura/escritura a nivel de disco físico. La base de datos BancoDigital, para realizar una escritura (naranja) presenta el crecimiento más alto, acumulando hasta aproximadamente 16.7 minutos de espera total de I/O de escritura hacia el final de la prueba, mientras que las esperas de lectura y las de las bases de datos del sistema (model_msdb, model_replicatedmaster) permanecen cercanas a 0. Esto significa que las operaciones

CRUD con proporción significativa de escrituras: create, update, delete, lo cual es coherente la mayor parte de la latencia a nivel de disco proviene específicamente de las operaciones de escritura sobre la base de datos de la aplicación, no de operaciones de lectura ni de procesos internos del motor de base de datos.

Figura 19

Gráfica de latencia del disco del aplicativo sincrónico



4.5.1.1.5 Estadística descriptiva de la prueba de carga del aplicativo sincrónico

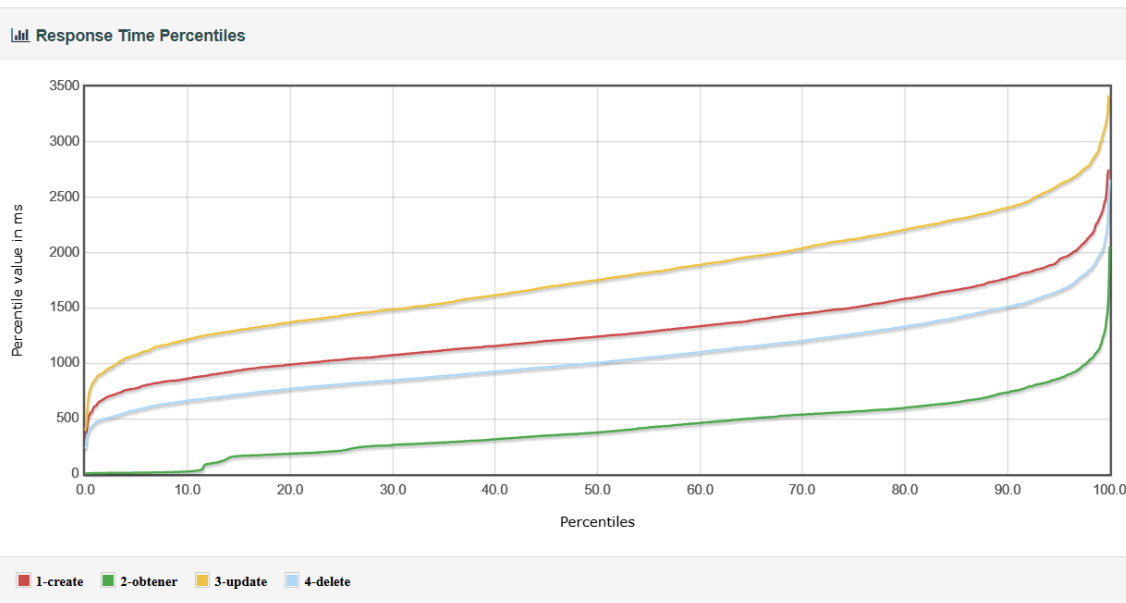
En la siguiente figura 20 se muestra el tiempo de respuesta acumulado por percentil para cada operación CRUD:

- Create (rojo):** muestra una mediana de aproximadamente 1150 ms y un percentil 90 de aproximadamente 1650 ms.
- Read(verde):** mantiene los valores más bajos en todo el rango, con un percentil 50 (mediana) de aproximadamente 250-300 ms y un percentil 90 cercano a 550-600 ms.
- Update (amarillo):** presenta los valores más altos en todos los percentiles, con una mediana de aproximadamente 1750 ms y un percentil 90 cercano a 2300 ms.
- Delete (celeste):** presenta una mediana de aproximadamente 900 ms y un percentil 90 cercano a 1400 ms.

En el extremo superior percentil 99-100, las cuatro curvas muestran un incremento, alcanzando valores máximos de aproximadamente 3400 ms para update, 2750 ms para create, 2050 ms para delete y 2000 ms para obtener, lo que evidencia que, existe un subconjunto de solicitudes con tiempos de respuesta considerablemente más altos que el resto.

Figura 20

Gráfica de percentiles del aplicativo sincrónico



Validación de los resultados

a) Tasa de error

Se registró una tasa de error del 0.00% en las cuatro operaciones evaluadas sobre un total de 26 425 mediciones, lo que indica que el aplicativo no presentó fallos durante toda la prueba, sin fallos, excepciones ni interrupciones de conexión registradas en ninguna de las operaciones.

- a) **Create:** 6612 ejecuciones, 0 fallos (0.00%)
- b) **Read:** 6608 ejecuciones, 0 fallos (0.00%)
- c) **Update:** 6607 ejecuciones, 0 fallos (0.00%)
- d) **Delete:** 6598 ejecuciones, 0 fallos (0.00%)

b) Error estándar de la media

El error estándar de la media ($SE = \text{desviación estándar} / \sqrt{n}$) resulta pequeño en relación con el promedio de cada operación, lo que indica que los valores constituyen estimaciones precisas y confiables del verdadero comportamiento del sistema, esto mide cuando variaría el promedio si repitiera nuevamente la prueba.

- a) **Create:**
 $SE = 359.61 / \sqrt{6612} = 4.42 \text{ ms sobre un promedio de } 1288 \text{ ms} = 0.34\%$
- b) **Read:**
 $SE = 261.11 / \sqrt{6608} = 3.21 \text{ ms sobre un promedio de } 405.81 \text{ ms} = 0.79\%$
- c) **Update:**

$$SE = 473.59 / \sqrt{6607} = 5.83 \text{ ms sobre un promedio de } 1787.86 \text{ ms} = 0.33\%$$

d) **Delete:**

$$SE = 335.00 / \sqrt{6598} = 4.12 \text{ ms sobre un promedio de } 1054.62 \text{ ms} = 0.39\%$$

c) Cercanía entre promedio y mediana

Se evidencia una cercanía similar entre el promedio y la mediana en las cuatro operaciones evaluadas, con diferencias minimas en todos los casos, lo que confirma que los resultados no se encuentran distorsionados por valores extremos y que la distribución de tiempos de respuesta es relativamente simétrica.

- a) **Create:** promedio 1288 ms frente a mediana 1242 ms (diferencia de 3.6%)
- b) **Read:** promedio 405.81 ms frente a mediana 377 ms (diferencia de 7.1%)
- c) **Update:** promedio 1787.86 ms frente a mediana 1752 ms (diferencia de 2.0%)
- d) **Delete:** promedio 1054.62 ms frente a mediana 1007 ms (diferencia de 4.5%)

d) Progresión gradual de percentiles

En las cuatro operaciones se observa un incremento progresivo y sin saltos abruptos entre los percentiles 50, 90, 95 y 99, lo que descarta la presencia de comportamientos erráticos o bimodales en el sistema durante la prueba.

- a) **Create:** mediana 1242 ms, → 90% 1770 ms, → 95% 1931 ms, → 99% 2314 ms
- b) **Read:** mediana 377 ms, → 90% 737 ms, → 95% 863 ms, → 99% 1134 ms
- c) **Update:** mediana 1752 ms, → 90% 2405 ms, → 95% 2614 ms, → 99% 2955 ms
- d) **Delete:** mediana 1007 ms, → 90% 1509 ms, → 95% 1653 ms, → 99% 1974 ms

e) Throughput similar entre operaciones

El throughput se mantuvo prácticamente idéntico entre las cuatro operaciones, lo que evidencia un procesamiento predecible y balanceado del sistema bajo la carga aplicada, independientemente del tiempo de respuesta individual de cada operación.

- a) **Create:** 3.70 transacciones/seg
- b) **Read:** 3.71 transacciones/seg
- c) **Update:** 3.70 transacciones/seg
- d) **Delete:** 3.70 transacciones/seg

f) Volumen y duración suficientes para diluir el ruido de corto plazo

El volumen total de mediciones recolectadas durante los 30 minutos de prueba continua resulta suficiente para que efectos puntuales de corta duración.

- a) **Create:** 6612 mediciones
- b) **Read:** 6608 mediciones
- c) **Update:** 6607 mediciones
- d) **Delete:** 6598 mediciones

Total: 26 425 mediciones en 30 minutos de prueba continua

4.5.1.2 Resultados del aplicativo asincrónico

Los resultados que se tuvo para el aplicativo asincrónico fueron los siguientes:

Tabla 8

Resultados del aplicativo asincrónico en la prueba de carga

Endpoint	Cantida d de ejecucio nes	Tiempo de respuest a promedi o (ms)	Tiemp o mínim o de respues ta (ms)	Tiemp o máxim o de respues ta (ms)	Desvia ción estánd ar	Through put	Media na	90%	95%
Creación	7884	1204	336	3111	323.69	4.4/sec.	1165	1617	1793
Obtener	7875	342	7	2404	178.51	4.4/sec.	301	580	683
Edición	7872	1138	195	3831	404.94	4.4/sec.	1097	1674	1854
Eliminación	7866	1119	251	3066	345.89	4.4/sec.	1072	1582	1766
Total	31497	957	7	3831	479.06	17.6/sec.	967	1551	1731

Nota. Elaboración propia.

Se detalla a continuación sobre cada endpoint sus métricas:

- a) **Resultado del endpoint Creación:** Se ejecutó 7884 veces, con un promedio de respuesta de 1204 ms, el mínimo tiempo de respuesta fue de 336 ms y el máximo de 3111 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 323.69, con un throughput de 4.4/sec., con una mediana de 1165 y percentil 90 de 1617 y percentil 95 de 1793 en JMeter.

- b) Resultado del endpoint Obtener:** Se ejecutó 7875 veces, con un promedio de respuesta de 342 ms, el mínimo tiempo de respuesta fue de 7 ms y el máximo de 2404 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 178.51, con un throughput de 4.4/sec., con una mediana de 301 y percentil 90 de 580 y percentil 95 de 683 en JMeter.
- c) Resultado del endpoint Edición:** Se ejecutó 7872 veces, con un promedio de respuesta de 1138 ms, el mínimo tiempo de respuesta fue de 195 ms y el máximo de 3831 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 404.94, con un throughput de 4.4/sec., con una mediana de 1097 y percentil 90 de 1674 y percentil 95 de 1854 en JMeter.
- d) Resultado del endpoint Eliminación:** Se ejecutó 7866 veces, con un promedio de respuesta de 1119 ms, el mínimo tiempo de respuesta fue de 251 ms y el máximo de 3066 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 345.89, con un throughput de 4.4/sec., con una mediana de 1072 y percentil 90 de 1582 y percentil 95 de 1766 en JMeter.

4.5.1.2.1 Latencia

a) Latencia durante la prueba

En la siguiente figura 21 se muestra la latencia promedio en el tiempo transcurrido de la prueba para las 4 operaciones CRUD. El gráfico muestra latencia promedio vs el tiempo de la prueba, en la cual mide específicamente el tiempo hasta recibir el primer byte de respuesta del servidor.

Se observa un comportamiento similar para cada operación donde hay una fase de ascenso esto sería normal ya que en los primeros minutos de la prueba específicamente 5 minutos va creciendo los usuarios hasta llegar a los 20 usuarios, seguida de una fase de estable de 20 minutos y un descenso en los últimos 5 minutos ya que los usuarios están disminuyendo para terminar la prueba.

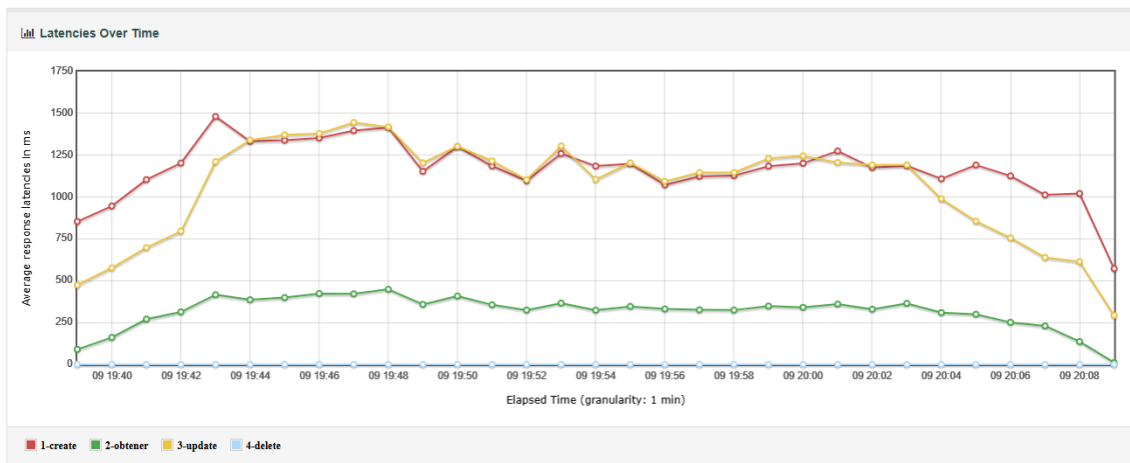
Por operación se detalla lo siguiente:

- a) Create (rojo):** Se ubica en un rango intermedio, estabilizándose entre 1000 a 1200ms con un pico máximo de 1482 en el minuto 19:43
- b) Read (verde):** Se muestra una latencia considerablemente menor con respecto a los endpoint de create y update, estabilizándose entre 300 a 400ms, esto normal ya que las operaciones de lectura son más rápidas.

- c) **Update (amarillo):** Presenta una latencia moderada y estable, estabilizándose entre 1100 - 1300ms, con un pico de máximo de 1446 en el minuto 19:47
- d) **Delete (azul):** Se mantiene en 0ms durante la prueba ya que para esta operación no se espera ni una respuesta(response) que tenga contenido, es por eso que la respuesta no tiene un byte para poder calcularlo.

Figura 21

Gráfica de latencia del aplicativo asincrónico



4.5.1.2.2 Response Time

a) Tiempo de respuesta durante la prueba

En el siguiente gráfico 22 se muestra el tiempo de respuesta promedio en el tiempo que duro la prueba, se puede observar el grafico de las 4 operaciones (CRUD). El tiempo de respuesta mide el proceso completo de la solicitud, desde el envío hasta la recepción total de la respuesta.

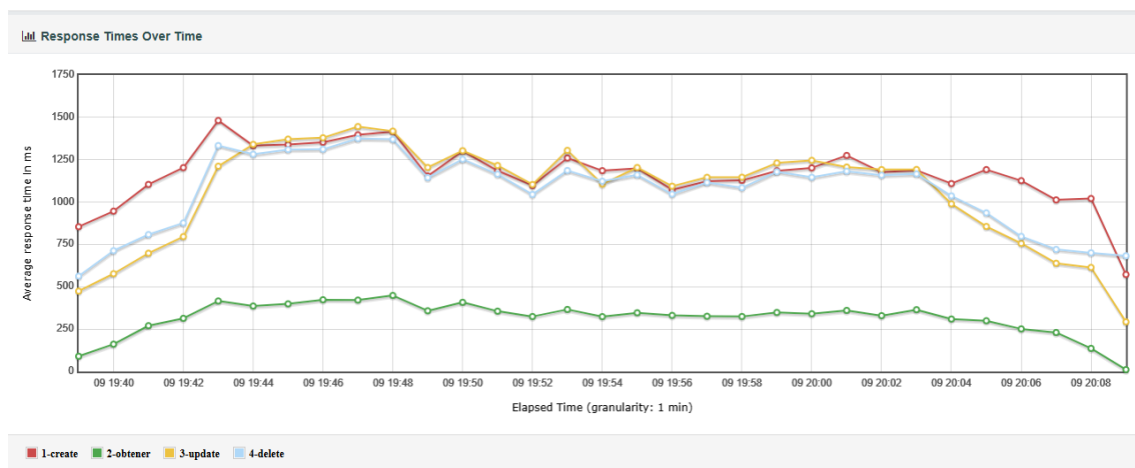
En el gráfico se identifican también 3 fases que la primera es la de creciente o ascenso en la cual se puede observar que en los primeros minutos hasta el minuto 5 se muestra una elevación ya que en esos momentos se están incrementando los usuarios hasta llegar hasta los 20 usuarios, después se mantiene una estabilidad durante los 20 minutos ya que se encuentra activos los 20 usuarios, después una fase de descenso en la cual los usuarios van decreciendo para terminar la prueba. Se detalla por operación:

- a) **Create (rojo):** Se ubica en un rango de 1100 a 1300, con el punto más alto en 1482 en el tiempo 19:43. Estos comportamientos es propio de una operación de escritura que requiere insertar registros a las tablas de cuenta, saldo y operación en todo el proceso.

- b) **Read (verde):** Es la operación más rápida, ya que oscila entre 300 a 450 durante toda la fase. Esto es rápido ya que no bloquea las tablas al momento de consultar. Teniendo un punto máximo de 449 en el tiempo 19:48.
- c) **Update (amarillo):** Presentan el tiempo de respuesta estable como los otros endpoint, estabilizándose entre 1100 y 1300ms, con su pico máximo de 1446ms en el tiempo 19:47 de la prueba, esto debido que busca los registros en cada tabla, realiza la modificación y guardado del registro en las tablas teniendo una mayor sobrecarga transaccional.
- d) **Delete(azul):** Es el proceso que se ubica en una estabilización de 1000 a 1200, un valor estable ya que se encuentra en tiempos regulares iguales que create y update ya que este proceso se encarga de buscar y eliminar el registro, teniendo un tiempo de respuesta más alto en 1375 en el tiempo 19:47.

Figura 22

Gráfica de tiempo de respuesta durante la prueba



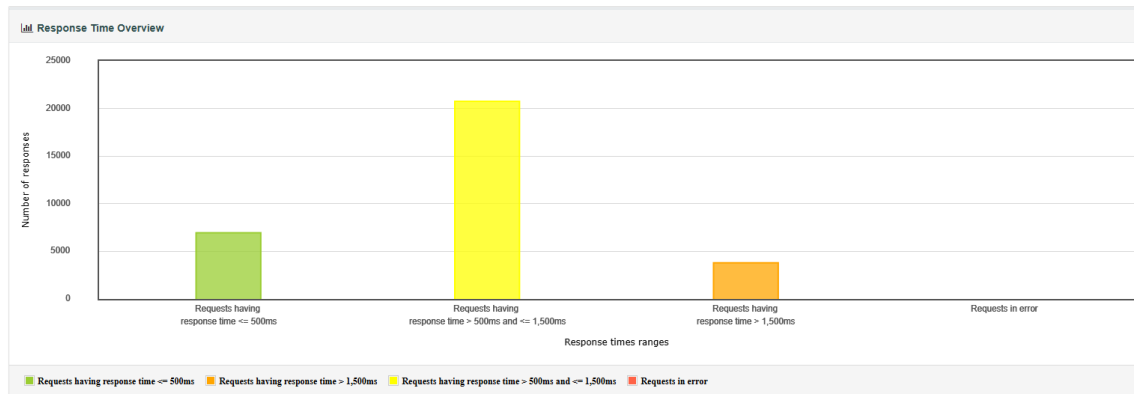
b) Distribución de cantidad de peticiones y tiempo de respuesta

En la siguiente figura 23 se muestra la distribución general de cantidades de respuestas, agrupando según el rango de tiempo que duro la peticiones se dividen en 3 partes:

- a) Tiempo de respuesta de ≤ 500 ms: Se 6949 peticiones
- b) Tiempo de respuesta entre $500 < > 1500$: 20758 peticiones
- c) Tiempo de respuesta mayor de >1500 : 3790 peticiones

Figura 23

Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico



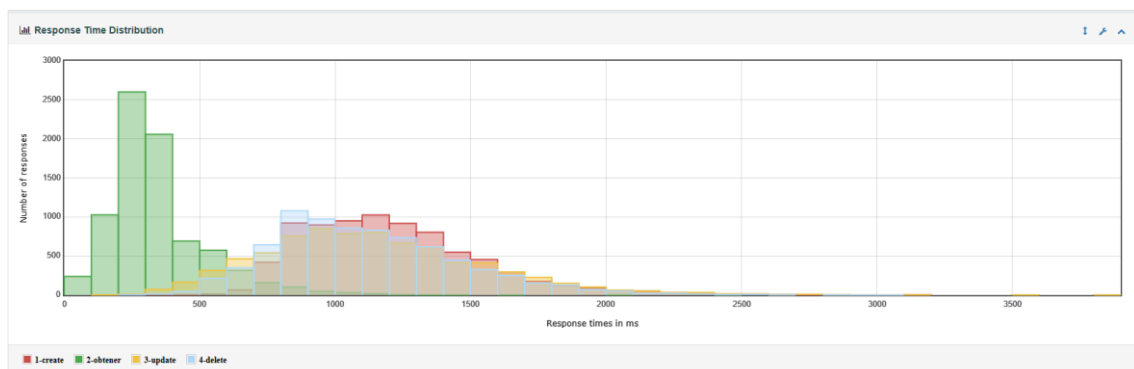
c) Distribución de tiempo de respuesta

En la siguiente figura 24 se muestra la distribución de frecuencias de tiempo de respuesta, separada por operación CRUD.

- a) **Create (rojo):** Se distribuye en mayo concentración en 800 a 1200ms, teniendo la mayor distribución en el rango de 1025 en 1100 a 1200ms.
- b) **Read (verde):** Varias peticiones se concentran en el rango de 0 a 500ms, teniendo en la mayor distribución de 2602 peticiones en 200 a 300 ms.
- c) **Update (amarillo):** Se distribuye en mayor concentración en 800 a 1300ms, teniendo la mayor distribución con 852 peticiones en 900 a 1000ms.
- d) **Delete(azul):** Se distribuye principalmente en 800 a 1300ms, teniendo en la mayoría 1079 peticiones en 800 a 900ms.

Figura 24

Gráfica de distribución de tiempo de respuesta



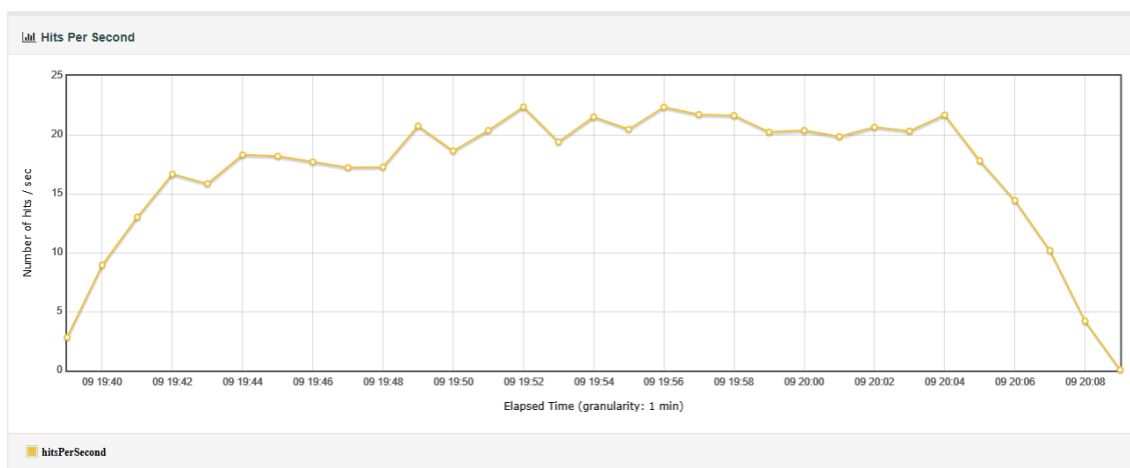
4.5.1.2.3 Throughput

a) TPS en general

En la siguiente figura 25 se puede observar el total de los TPS en el transcurso de la prueba como va variando en todo el transcurso, en este caso es la suma general de las 4 operaciones CRUD, se puede observar que hay un ascenso cuando empieza en los primeros minutos es normal ya que varios usuarios están siendo activados para realizar peticiones y dentro de los 20 minutos se mantiene estable manteniendo un rango de 18 a 22 TPS y teniendo un pico máximo de 22.37 en el tiempo 19:52 y en los últimos 5 minutos teniendo una descenso ya que usuarios se van disminuyendo.

Figura 25

Gráfica de TPS en general del aplicativo asincrónico



b) TPS por operación

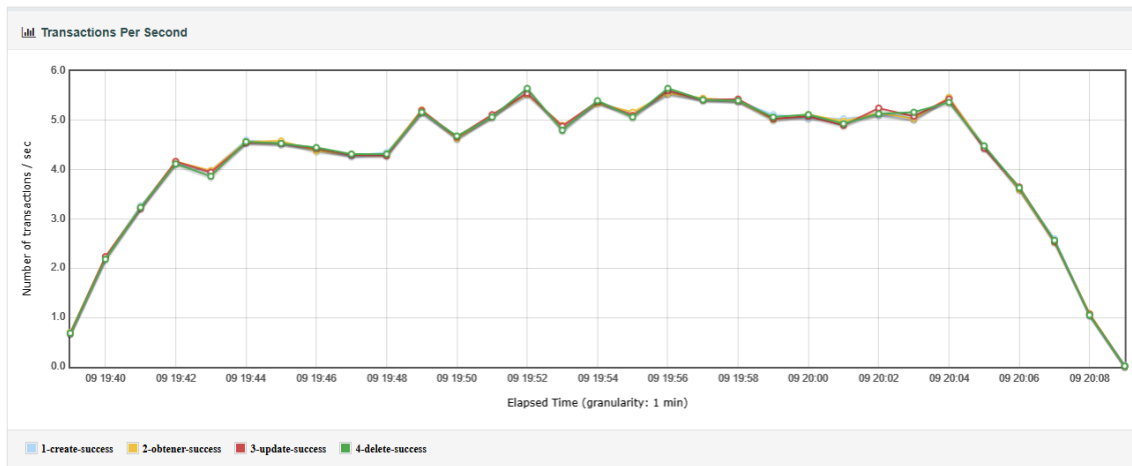
En la siguiente figura 26 se muestra el throughput (transacciones por segundo) de las cuatro operaciones CRUD a lo largo del tiempo de prueba. Acá se observa que las 4 operaciones se comportan de manera prácticamente idénticas.

- a) **Create (azul):** Se puede observar que alcanza su punto máximo de 5.6 en varios tiempos 19:52, 19:56 y 20:04.
- b) **Read (amarillo):** Se puede observar que alcanza su punto máximo de 5.6 en varios tiempos 19:52, 19:56, 20:04.
- c) **Update (rojo):** Se puede observar que alcanza su punto máximo de 5.6 en dos tiempos 19:52, 19:56.

- d) **Delete(verde):** Se puede observar que alcanza su punto máximo de 5.7 en dos tiempos de 19.52, 19.56.

Figura 26

Gráfica del TPS por operación del aplicativo asincrónico



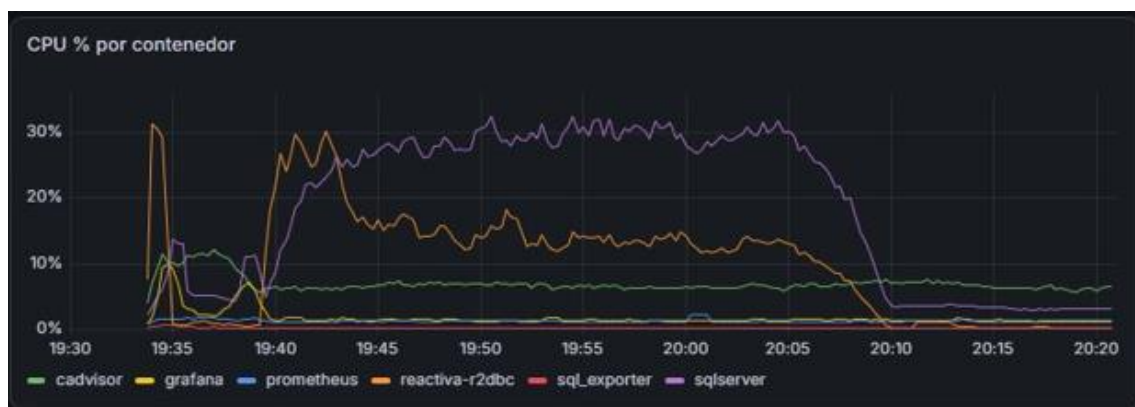
4.5.1.2.4 Consumo de recurso

a) Consumo de CPU

En la siguiente figura 27 se observa el consumo de CPU por contenedor Docker durante los 30 minutos de prueba. El contenedor de SQL Server (morado) presenta el consumo más alto, alcanzando picos de aproximadamente 30-32% durante la fase estable (19:52-19:54), mientras que el contenedor de la aplicación reactivo (naranja) se mantiene en un rango de 13-17%, con un pico de 30% al inicio de la prueba (19:42). Los contenedores de monitoreo (cAdvisor, Grafana, Prometheus) presentan un consumo pequeño, por debajo del 5%, confirmando que no interfieren en la medición.

Figura 27

Gráfica del consumo de CPU del aplicativo asincrónico

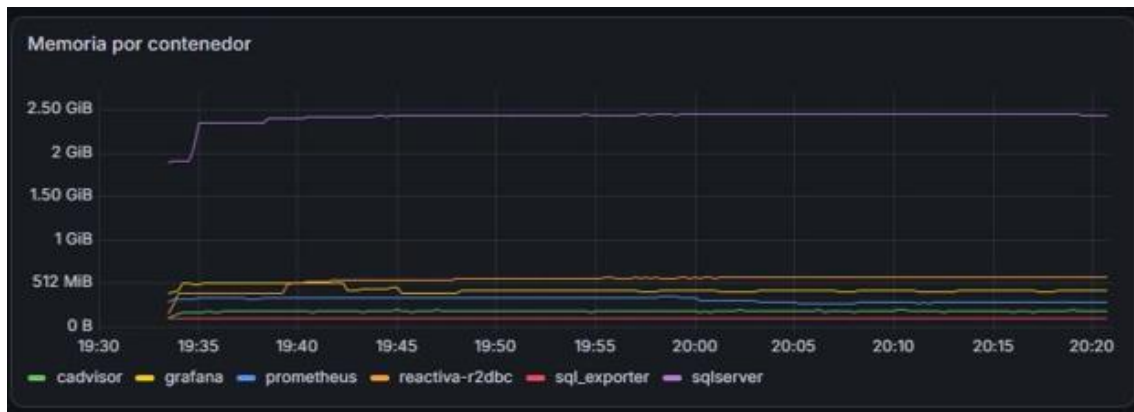


b) Consumo de memoria

En la siguiente figura 28 se muestra la memoria se mantiene prácticamente constante durante toda la prueba, sin picos ni variaciones asociadas al aumento de carga. El contenedor de SQL Server (morado) consume de forma estable 2.5 GiB, mientras que la aplicación reactiva (naranja) se mantiene alrededor de 400-512 MiB.

Figura 28

Gráfica de consumo de memoria del aplicativo asincrónico



c) Consumo de JVM

1 JVM Heap Memory

En la siguiente figura 29 se muestra el uso del heap de la JVM de manera desagregado por generación: G1 Eden Space, G1 Old Gen y G1 Survivor Space, junto con el límite máximo configurado. Los tres componentes se mantienen en valores estables durante toda la prueba, lo que indica que la aplicación está operando normal sin alcanzar su capacidad máxima de heap asignada. Esto significa que, para esta carga de prueba con 20 usuarios concurrentes, la memoria no constituye un factor que limite el rendimiento observado, y que existe un amplio margen antes de que la gestión de memoria se convierta en un cuello de botella.

Figura 29

Gráfica de JVM Heap Memory del aplicativo asincrónico

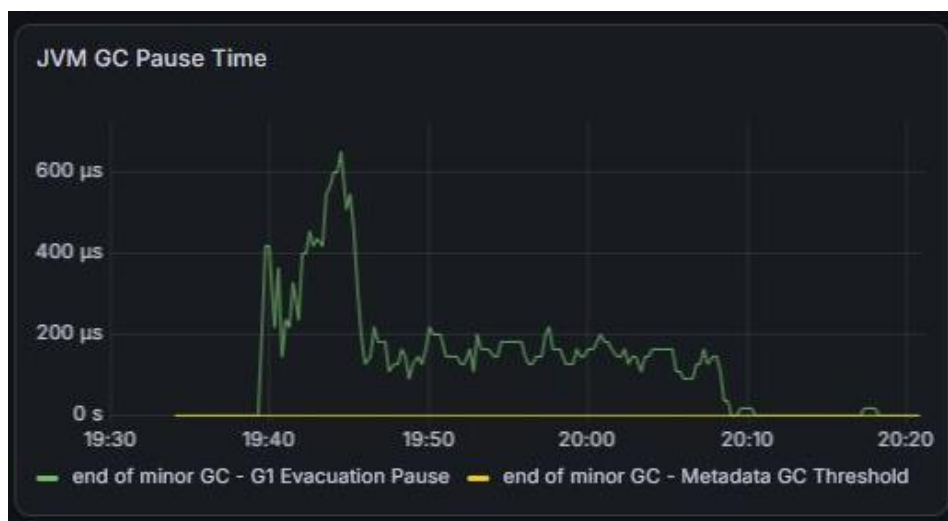


2 JVM GC Pause Time

En la siguiente figura 30 se observa el tiempo de pausa generado por el recolector de basura (Garbage Collector) durante la ejecución. Las pausas se mantienen mayormente por debajo de 500 microsegundos a lo largo de la prueba, con un pico de aproximadamente 600 microsegundos alrededor del minuto 19:45, coincidiendo con el inicio de la prueba. El comportamiento del GC es regular y de bajo impacto, lo que indica que la gestión de memoria de la JVM no representa una fuente significativa de latencia adicional para las operaciones CRUD evaluadas.

Figura 30

Gráfica de JVM GC Pause Time del aplicativo asincrónico



3 JVM Threads

En la siguiente figura 31 se muestra el número de hilos vivos (verde) e hilos daemon (amarillo) de la JVM. Se observa un incremento desde 12 hilos en reposo

hasta estabilizarse entre 28-30 hilos activos durante la fase de carga en el tiempo (19:45-20:20), y luego un descenso a los valores base al finalizar la prueba. Este comportamiento es el esperado, donde la petición HTTP entrante ocupa un hilo dedicado durante toda su ejecución de procesamiento + espera de respuesta de la base de datos. El número de hilos supera ese valor ya que tiene valor de 28-30 esto refleja un adicional de hilos internos del framework y del contenedor de aplicaciones Tomcat y no solo los hilos dedicados directamente a las peticiones de usuario.

Figura 31

Gráfica de JVM Threads del aplicativo asincrónico



d) Conexiones R2dbc

En la siguiente figura 32 se muestra el estado del pool de conexiones a la base de datos, conexiones activas (verde), idle/libres (amarillo) y en espera/pending (azul). Se observa que, al iniciar la carga en el tiempo 19:40, las conexiones activas alcanzan rápidamente su máximo configurado de 15 conexiones antes de 19:45 y se mantienen en ese tope durante toda la fase, mientras que simultáneamente aparecen conexiones en espera (pending), estabilizándose en aproximadamente entre 5 a 10. Esto indica que el pool de conexiones se saturó bajo la carga de 20 usuarios concurrentes, y que existían peticiones adicionales esperando a que se liberé una conexión antes de poder ejecutar la siguiente operación contra la base de datos generando un cuello de botella directamente y esto es atribuible al tamaño configurado del pool, pero lo que se observa es que hay hasta 13 conexiones en espera eso significa que se liberan más rápido con respecto con la

imperativa por el mismo de ser más flexible y de liberar más rápido los puntos de conexión.

Figura 32

Gráfica de conexiones de R2dbc del aplicativo asincrónico



e) Throughput SQL Server

En la siguiente figura 33 se muestran dos métricas batch requests/sec (verde), que mide lotes de comandos SQL ejecutados, y transacciones/sec (amarillo). Ambas curvas muestran un crecimiento sostenido desde el inicio de la prueba hasta estabilizarse alrededor del minuto 19:40, alcanzando aproximadamente 750K batch requests acumulados y 100K transacciones acumuladas.

Figura 33

Gráfica de Throughput del consumo de sql server del aplicativo asincrónico



f) Locks y deadlocks

En la siguiente figura 34 se observa un crecimiento sostenido de los lock waits durante la fase de carga, alcanzando aproximadamente 12.3K esperas acumuladas antes de

estabilizarse, mientras que la línea de deadlocks se mantiene prácticamente en 0 durante toda la prueba. Esto indica que, si existieron esperas por bloqueos de fila/tabla y no se registraron deadlocks que hubieran obligado a SQL Server a abortar transacciones, lo cual es confiable con la tasa de error del 0% observada en JMeter.

Figura 34

Gráfica de lock y deadlock del aplicativo asincrónico

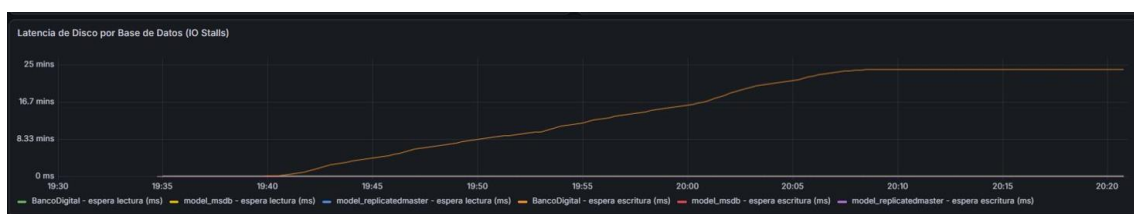


g) Latencia de Disco por base de datos

En la siguiente figura 35 se mide el tiempo acumulado de espera en operaciones de lectura/escritura a nivel de disco físico. La base de datos BancoDigital, para realizar una escritura (naranja) presenta el crecimiento más alto, acumulando hasta aproximadamente 24.5 minutos de espera total de I/O de escritura hacia el final de la prueba, mientras que las esperas de lectura y las de las bases de datos del sistema (model_msdb, model_replicatedmaster) permanecen cercanas a 0. Esto significa que las operaciones CRUD con proporción significativa de escrituras: create, update, delete, lo cual es coherente la mayor parte de la latencia a nivel de disco proviene específicamente de las operaciones de escritura sobre la base de datos de la aplicación, no de operaciones de lectura ni de procesos internos del motor de base de datos.

Figura 35

Gráfica de latencia del disco del aplicativo asincrónico



4.5.1.2.5 Estadística descriptiva de la prueba de carga del aplicativo asincrónico

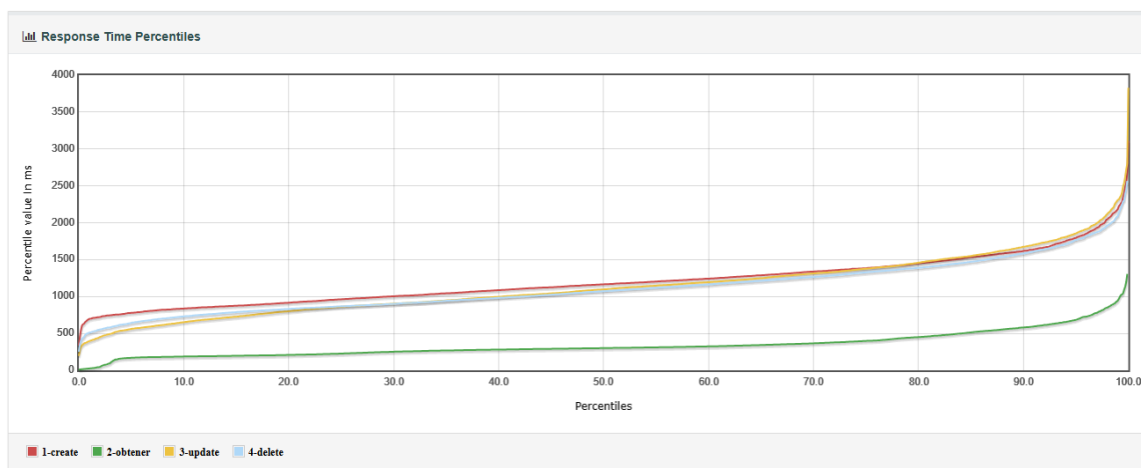
En la siguiente figura 36 se muestra el tiempo de respuesta acumulado por percentil para cada operación CRUD:

- a) **Create (rojo):** muestra una mediana de aproximadamente 1165 ms y un percentil 90 de aproximadamente 1617 ms.
- b) **Read(verde):** mantiene los valores más bajos en todo el rango, con un percentil 50 (mediana) de aproximadamente 301 ms y un percentil 90 cercano a 579 ms.
- c) **Update (amarillo):** presenta los valores más altos en todos los percentiles, con una mediana de aproximadamente 1098 ms y un percentil 90 cercano a 1674 ms.
- d) **Delete (celeste):** presenta una mediana de aproximadamente 1071 ms y un percentil 90 cercano a 1581 ms.

En el extremo superior percentil 99-100, las cuatro curvas muestran un incremento, alcanzando valores máximos de aproximadamente 2310 ms para update, 2226 ms para create, 2093 ms para delete y 947 ms para obtener, lo que evidencia que, existe un subconjunto de solicitudes con tiempos de respuesta considerablemente más altos que el resto.

Figura 36

Gráfica de percentiles del aplicativo asincrónico



Validación de los resultados

a) Tasa de error

Se registró una tasa de error del 0.00% en las cuatro operaciones evaluadas sobre un total de 31 497 mediciones, lo que indica que el aplicativo no presentó fallos durante toda la

prueba, sin fallos, excepciones ni interrupciones de conexión registradas en ninguna de las operaciones.

- a) **Create:** 7884 ejecuciones, 0 fallos (0.00%)
- b) **Read:** 7875 ejecuciones, 0 fallos (0.00%)
- c) **Update:** 7872 ejecuciones, 0 fallos (0.00%)
- d) **Delete:** 7866 ejecuciones, 0 fallos (0.00%)

b) Error estándar de la media

El error estándar de la media ($SE = \text{desviación estándar} / \sqrt{n}$) resulta pequeño en relación con el promedio de cada operación, lo que indica que los valores constituyen estimaciones precisas y confiables del verdadero comportamiento del sistema, esto mide cuando variaría el promedio si repitiera nuevamente la prueba.

- a) **Create:**
 - a. $SE = 323.69 / \sqrt{7884} = 3.64$ ms sobre un promedio de 1204 ms = 0.30%
- b) **Read:**
 - a. $SE = 178.51 / \sqrt{7875} = 2.01$ ms sobre un promedio de 342 ms = 0.58%
- c) **Update:**
 - a. $SE = 404.94 / \sqrt{7872} = 4.56$ ms sobre un promedio de 1138 ms = 0.40%
- d) **Delete:**
 - a. $SE = 345.89 / \sqrt{7866} = 3.89$ ms sobre un promedio de 1119 ms = 0.34%

c) Cercanía entre promedio y mediana

Se evidencia una cercanía similar entre el promedio y la mediana en las cuatro operaciones evaluadas, con diferencias mínimas en todos los casos, lo que confirma que los resultados no se encuentran distorsionados por valores extremos y que la distribución de tiempos de respuesta es relativamente simétrica.

- a) **Create:** promedio 1204 ms frente a mediana 1165 ms (diferencia de 3.3 %)
- b) **Read:** promedio 342 ms frente a mediana 301 ms (diferencia de 9%)
- c) **Update:** promedio 1138 ms frente a mediana 1097 ms (diferencia de 3.6%)
- d) **Delete:** promedio 1119 ms frente a mediana 1072 ms (diferencia de 4.3%)

d) Progresión gradual de percentiles

En las cuatro operaciones se observa un incremento progresivo y sin saltos abruptos entre los percentiles 50, 90, 95 y 99, lo que descarta la presencia de comportamientos erráticos o bimodales en el sistema durante la prueba.

- a) **Create:** mediana 1165 ms, → 90% 1617 ms, → 95% 1793 ms, → 99% 2187ms
- b) **Read:** mediana 301 ms, → 90% 580 ms, → 95%, 684 ms, → 99% 947 ms
- c) **Update:** mediana 1097 ms, → 90% 1674 ms, → 95% 1854 ms, → 99% 2310 ms
- d) **Delete:** mediana 1072 ms, → 90% 1582 ms, → 95% 1766 ms, → 99% 2093 ms

e) Throughput similar entre operaciones

El throughput se mantuvo prácticamente idéntico entre las cuatro operaciones, lo que evidencia un procesamiento predecible y balanceado del sistema bajo la carga aplicada, independientemente del tiempo de respuesta individual de cada operación.

- a) **Create:** 4.42 transacciones/seg
- b) **Read:** 4.43 transacciones/seg
- c) **Update:** 4.42 transacciones/seg
- d) **Delete:** 4.42 transacciones/seg

f) Volumen y duración suficientes para diluir el ruido de corto plazo

El volumen total de mediciones recolectadas durante los 30 minutos de prueba continua resulta suficiente para que efectos puntuales de corta duración.

- a) **Create:** 7884 mediciones
- b) **Read:** 7875 mediciones
- c) **Update:** 7872 mediciones
- d) **Delete:** 7866 mediciones

Total: 31497 mediciones en 30 minutos de prueba continua

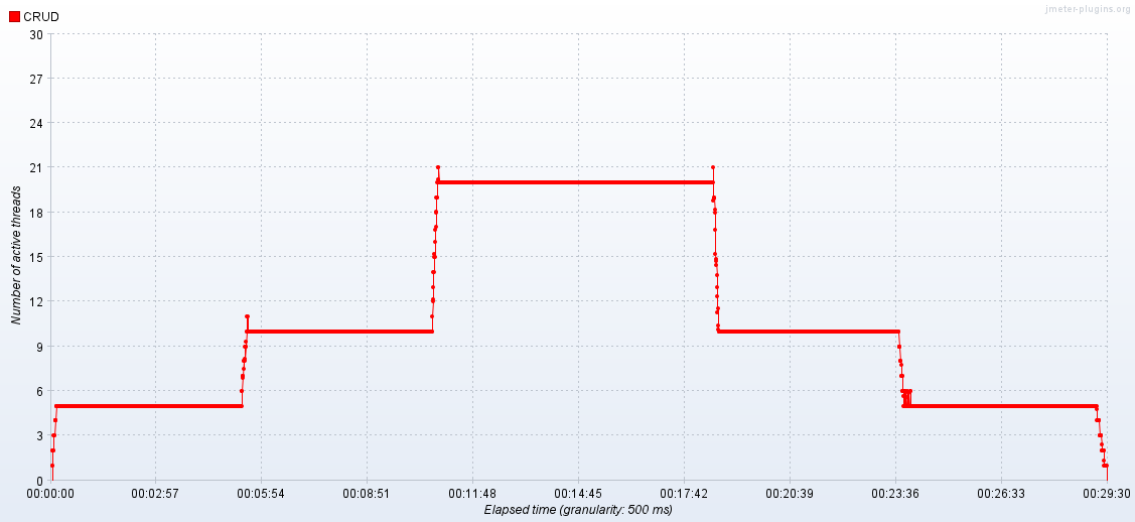
4.5.2 Pruebas de estrés

Como se muestra la figura 37, para esta prueba de estrés se ejecutaron 20 hilos (usuarios virtuales) con una duración de 30 minutos, de los cuales se comenzó con 0 usuario y aumentándose gradualmente llegando a 5 en los 10 s y en los siguientes 5 minutos aumentándose otros 5 usuarios después de otros minutos se aumentan otros 10 usuarios

hasta llegar a los 20 usuarios, teniendo así una totalidad de 20 usuarios por 8 minutos, y después en los últimos 5 minutos disminuyendo a 10 usuarios y en los otros siguientes minutos disminuyendo otros 10 hasta llegar al 0 usuario.

Figura 37

Prueba de estrés



4.5.2.1 Resultados del aplicativo sincrónico

Los resultados que se tuvo para el aplicativo sincrónico fueron los siguientes:

Tabla 9

Resultados del aplicativo sincrónico en la prueba de estrés

Endpoint	Cantida d de ejecucio nes	Tiempo de de respuest a promedi o (ms)	Tiemp o mínim o de respues ta (ms)	Tiemp o máxim o de respues ta (ms)	Desvia ción estánd ar	Through put	Media na	90%	95%
Creación	4989	1147	330	3129	349.74	2.8/sec.	1095	1609	1778
Obtener	4970	238	5	1717	265.72	2.8/sec.	169	615	757

Edición	4969	1507	474	4276	499.31	2.8/sec.	1425	2196	2431
Eliminación	4950	924	272	3008	336.53	2.8/sec.	868	1350	1547
Total	19878	954	5	4276	594.23	11.2/sec.	939	1708	1990

Nota. Elaboración propia.

Se detalla a continuación sobre cada endpoint sus métricas:

- a) **Resultado del endpoint Creación:** Se ejecutó 4989 veces, con un promedio de respuesta de 1147 ms, el mínimo tiempo de respuesta fue de 330 ms y el máximo de 3129 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 349.74, con un throughput de 2.8/sec., con una mediana de 1095 y percentil 90 de 1609 y percentil 95 de 1778 en JMeter.
- b) **Resultado del endpoint Obtener:** Se ejecutó 4970 veces, con un promedio de respuesta de 238 ms, el mínimo tiempo de respuesta fue de 5 ms y el máximo de 1717 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 265.72, con un throughput de 2.8/sec., con una mediana de 169 y percentil 90 de 615 y percentil 95 de 757 en JMeter.
- c) **Resultado del endpoint Edición:** Se ejecutó 4969 veces, con un promedio de respuesta de 1507 ms, el mínimo tiempo de respuesta fue 474 ms y el máximo de 4276 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 499.31, con un throughput de 2.8/sec., con una mediana de 1425 y percentil 90 de 2196 y percentil 95 de 2431 en JMeter.
- d) **Resultado del endpoint Eliminación:** Se ejecutó 4950 veces, con un promedio de respuesta de 924 ms, el mínimo tiempo de respuesta fue de 272 ms y el máximo de 3008 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 336.53, con un throughput de 2.8/sec., con una mediana de 868 y percentil 90 de 1350 y percentil 95 de 1547 en JMeter.

4.5.2.1.1 Latencia

- a) **Latencia durante la prueba**

En la siguiente figura 38 se muestra la latencia promedio en el tiempo transcurrido de la prueba para las 4 operaciones CRUD. El gráfico muestra latencia promedio vs el tiempo de la prueba, en la cual mide específicamente el tiempo hasta recibir el primer byte de respuesta del servidor.

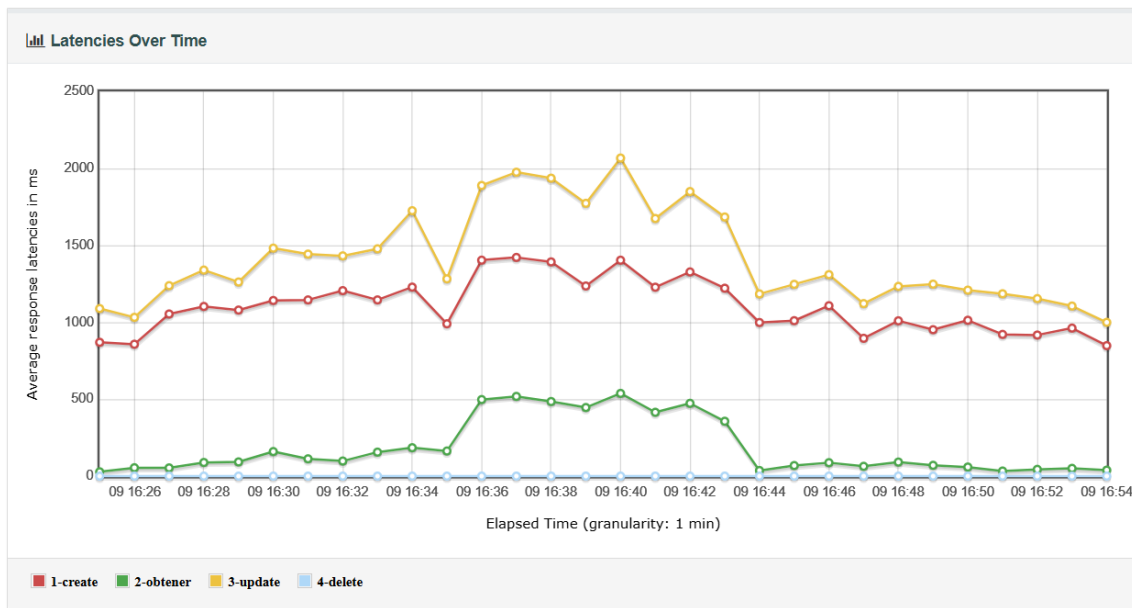
Se observa un comportamiento similar para cada operación donde hay una fase de ascenso esto sería normal ya que en los primeros minutos de la prueba específicamente 10 minutos va creciendo los usuarios hasta llegar a los 20 usuarios, seguida de una fase de estable de 8 minutos y un descenso en los últimos 12 minutos ya que los usuarios están disminuyendo para terminar la prueba.

Por operación se detalla lo siguiente:

- a) **Create (rojo):** Se ubica en un rango intermedio, estabilizándose entre 1000 a 1300 ms con un pico máximo de 1422 en el minuto 16:37.
- b) **Read (verde):** Se muestra una latencia considerablemente menor con respecto a los endpoint de create y update, estabilizándose entre 200 a 500ms, esto normal ya que las operaciones de lectura son más rápidas.
- c) **Update (amarillo):** Presenta una latencia más alta de las 4, estabilizándose entre 1500 - 1900ms, con un pico de máximo de 2060 en el minuto 16:40.
- d) **Delete (azul):** Se mantiene en 0ms durante la prueba ya que para esta operación no se espera ni una respuesta(response) que tenga contenido, es por eso que la respuesta no tiene un byte para poder calcularlo.

Figura 38

Gráfica de latencia del aplicativo sincrónico



4.5.2.1.2 Response Time

a) Tiempo de respuesta durante la prueba

En el siguiente gráfico 39 se muestra el tiempo de respuesta promedio en el tiempo que duro la prueba, se puede observar el grafico de las 4 operaciones (CRUD). El tiempo de respuesta mide el proceso completo de la solicitud, desde el envío hasta la recepción total de la respuesta.

En el gráfico se identifican también 3 fases que la primera es la de creciente o ascenso en la cual se puede observar que en los primeros minutos hasta el minuto 10 se muestra una elevación ya que en esos momentos se están incrementando los usuarios hasta llegar hasta los 20 usuarios, después se mantiene una estabilidad durante los 8 minutos ya que se encuentra activos los 12 usuarios, después una fase de descenso en la cual los usuarios van decreciendo para terminar la prueba. Se detalla por operación:

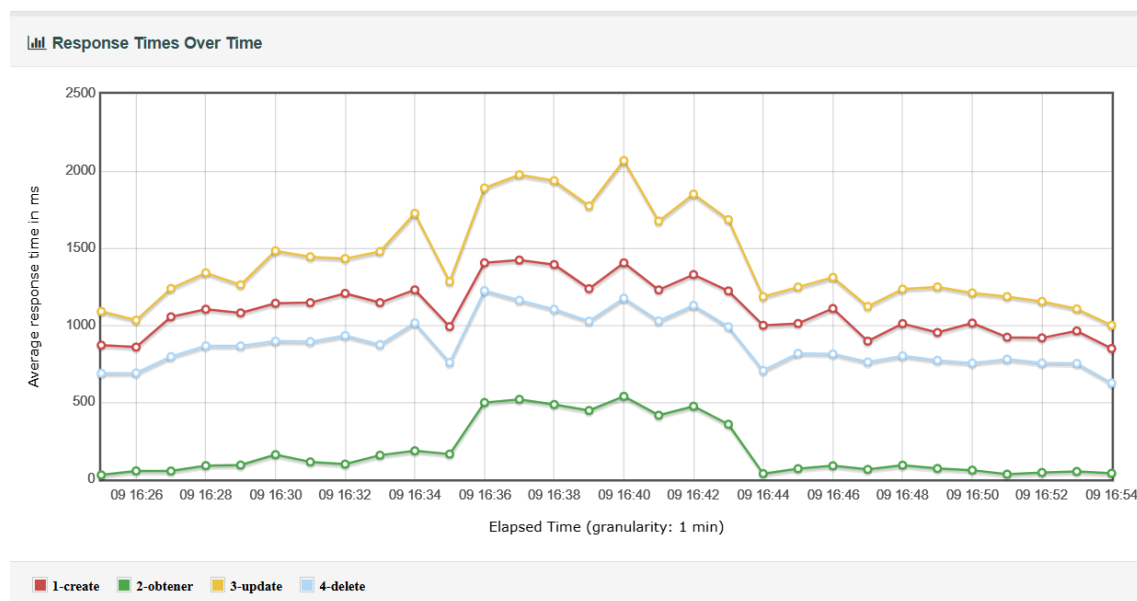
- Create (rojo):** Se ubica en un rango de 1000 a 1450, con el punto más alto en 1422 en el tiempo 16:37. Estos comportamientos es propio de una operación de escritura que requiere insertar registros a las tablas de cuenta, saldo y operación en todo el proceso.
- Read (verde):** Es la operación más rápida, ya que oscila entre 150 a 500 durante toda la fase. Esto es rápido ya que no bloquea las tablas al momento de consultar. Teniendo un punto máximo de 538ms en el tiempo 16:40.
- Update (amarillo):** Presentan el tiempo de respuesta más alto, estabilizándose entre 1500 y 2000ms, con su pico máximo de 2068ms en el tiempo 16:40 de la

prueba, esto debido que busca los registros en cada tabla, realiza la modificación y guardado del registro en las tablas teniendo una mayor sobrecarga transaccional.

- d) **Delete(azul)**: Es el proceso que se ubica en una estabilización de 800 a 1150, un valor estable ya que se encuentra en tiempos menores de créate y update ya que este proceso se encarga de buscar y eliminar el registro, teniendo un tiempo de respuesta más alto en 1224 en el tiempo 16:36.

Figura 39

Gráfica de tiempo de respuesta durante la prueba



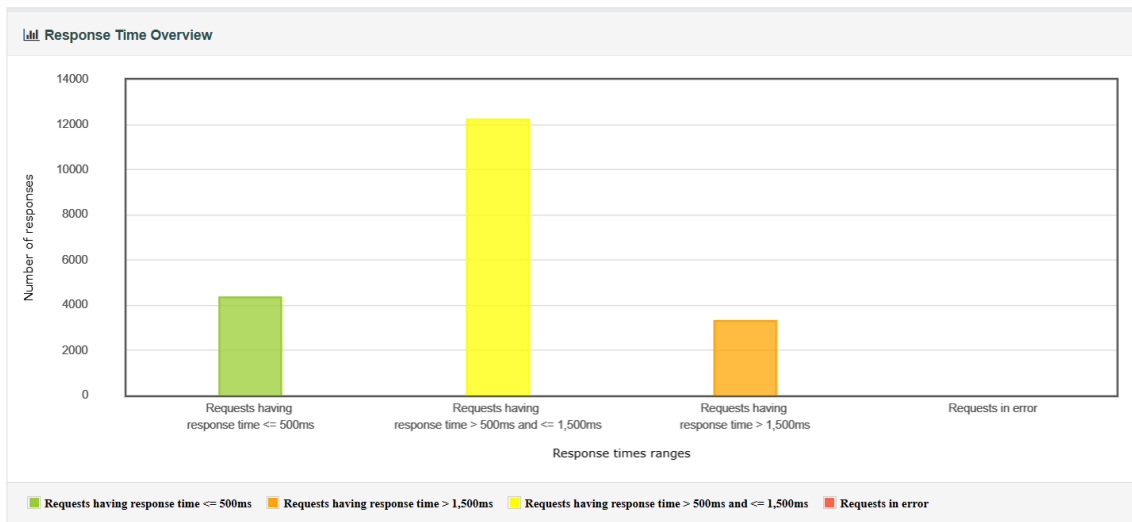
b) Distribución de cantidad de peticiones y tiempo de respuesta

En la siguiente figura 40 se muestra la distribución general de cantidades de respuestas, agrupando según el rango de tiempo que duro la peticiones se dividen en 3 partes:

- Tiempo de respuesta de ≤ 500 ms: Se 4345 peticiones
- Tiempo de respuesta entre $500 < > 1500$: 12237 peticiones
- Tiempo de respuesta mayor de >1500 : 3296 peticiones

Figura 40

Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico



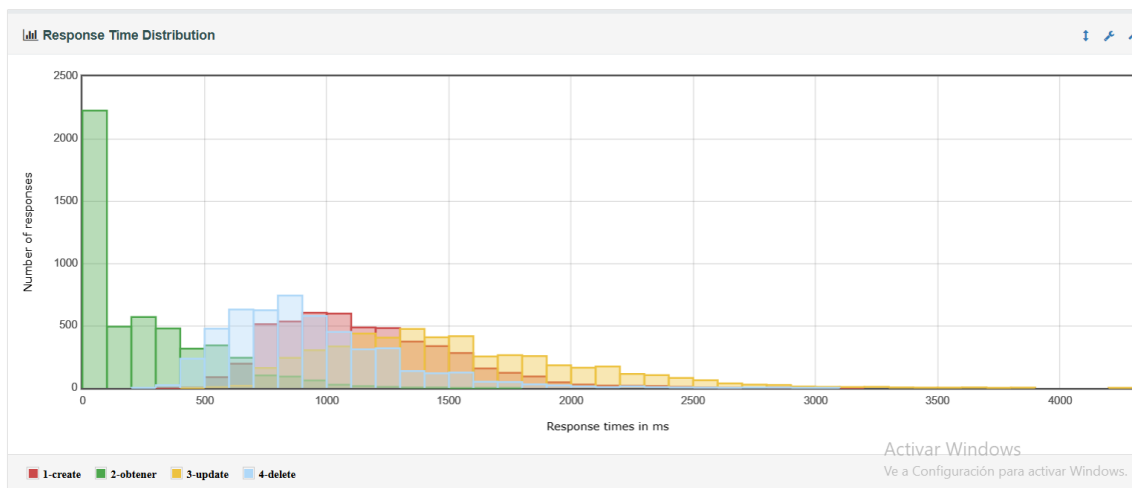
c) Distribución de tiempo de respuesta

En la siguiente figura 41 se muestra la distribución de frecuencias de tiempo de respuesta, separada por operación CRUD.

- a) **Create (rojo):** Se distribuye en mayo concentración en 700 a 1300ms, teniendo la mayor distribución en el rango de 603 en 900 a 1000ms.
- b) **Read (verde):** Varias peticiones se concentran en el rango de 0 a 700ms, teniendo en la mayor distribución de 2225 peticiones en 0 a 100 ms.
- c) **Update (amarillo):** Se distribuye en mayor concentración en 1100 a 1600ms, teniendo la mayor distribución con 406 peticiones en 1300 a 1400ms.
- d) **Delete(azul):** Se distribuye principalmente en 600 a 1000ms, teniendo en la mayoría 741 peticiones en 800 a 900ms.

Figura 41

Gráfica de distribución de tiempo de respuesta



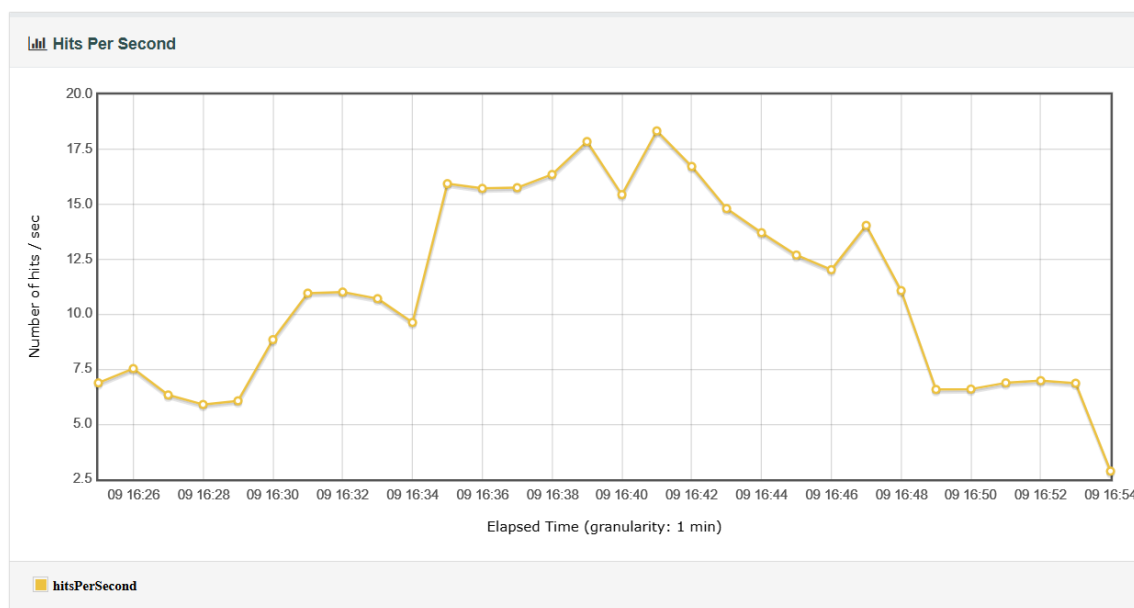
4.5.2.1.3 Throughput

a) TPS en general

En la siguiente figura 42 se puede observar el total de los TPS en el transcurso de la prueba como va variando en todo el transcurso, en este caso es la suma general de las 4 operaciones CRUD, se puede observar que hay un ascenso cuando empieza en los primeros minutos es normal ya que varios usuarios están siendo activados para realizar peticiones y dentro de los 8 minutos se mantiene estable manteniendo un rango de 16 a 17 TPS y teniendo un pico máximo de 18.35 en el tiempo 16:41 y en los últimos 12 minutos teniendo una descenso ya que usuarios se van disminuyendo.

Figura 42

Gráfica de TPS en general del aplicativo sincrónico



b) TPS por operación

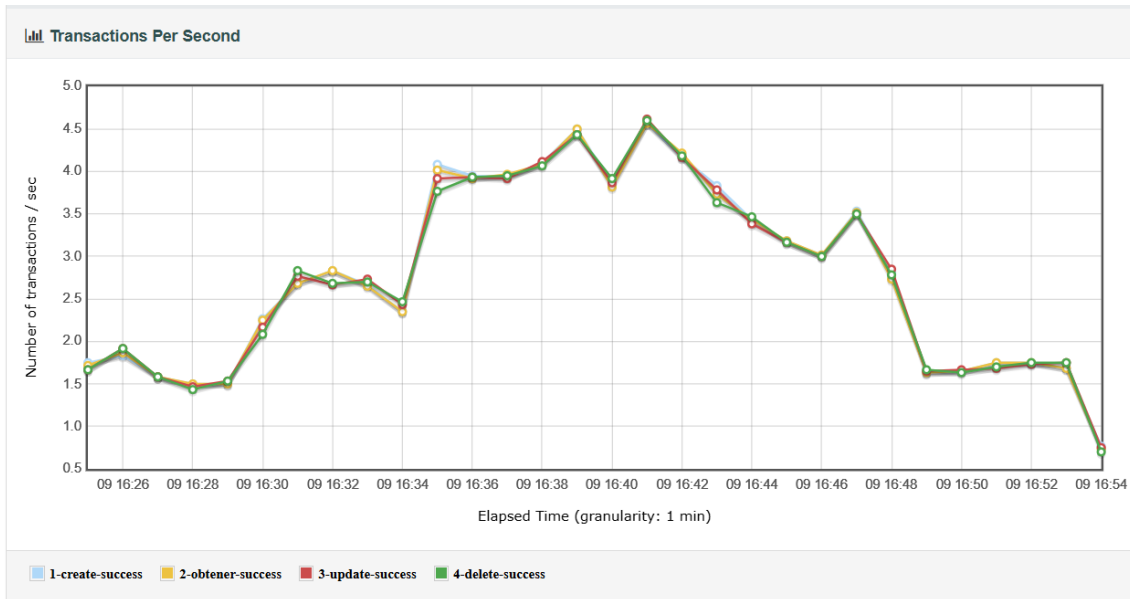
En la siguiente figura 43 se muestra el throughput (transacciones por segundo) de las cuatro operaciones CRUD a lo largo del tiempo de prueba. Acá se observa que las 4 operaciones se comportan de manera prácticamente identificas.

- a) **Create (azul):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 16:41.
- b) **Read (amarillo):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 16:41.

- c) **Update (rojo):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 16:41.
- d) **Delete(verde):** Se puede observar que alcanza su punto máximo de 4.6 en el tiempo de 16:41.

Figura 43

Gráfica del TPS por operación del aplicativo sincrónico



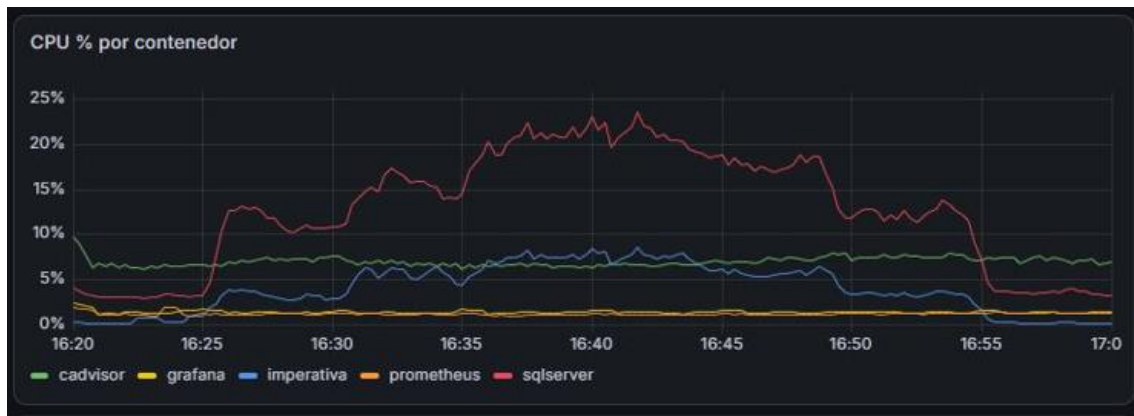
4.5.2.1.4 Consumo de recurso

a) Consumo de CPU

En la siguiente figura 44 se observa el consumo de CPU por contenedor Docker durante los 30 minutos de prueba. El contenedor de SQL Server (rojo) presenta el consumo más alto, alcanzando picos de aproximadamente 20-25% durante la fase estable (23:37-23:45), mientras que el contenedor de la aplicación imperativa (azul) se mantiene en un rango de 5-8%, con un pico de 8% al intermedio de la prueba (23:20). Los contenedores de monitoreo (cAdvisor, Grafana, Prometheus) presentan un consumo pequeño, por debajo del 5%, confirmando que no interfieren en la medición.

Figura 44

Gráfica del consumo de CPU del aplicativo sincrónico

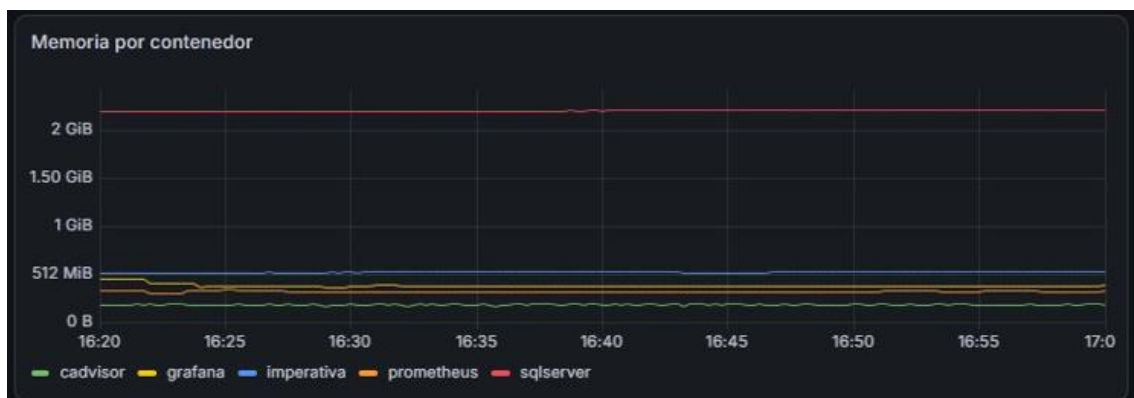


b) Consumo de memoria

En la siguiente figura 45 se muestra la memoria se mantiene prácticamente constante durante toda la prueba, sin picos ni variaciones asociadas al aumento de carga. El contenedor de SQL Server (rojo) consume de forma estable 2.3 GiB, mientras que la aplicación imperativa (azul) se mantiene alrededor de 512 MiB.

Figura 45

Gráfica de consumo de memoria del aplicativo sincrónico



c) Consumo de JVM

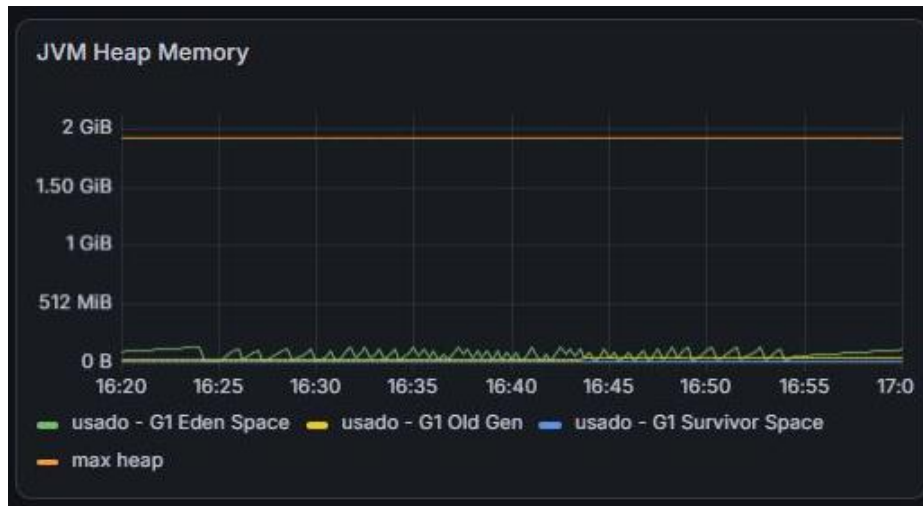
1) JVM Heap Memory

En la siguiente figura 46 se muestra el uso del heap de la JVM de manera desagregado por generación: G1 Eden Space, G1 Old Gen y G1 Survivor Space, junto con el límite máximo configurado. Los tres componentes se mantienen en valores estables durante toda la prueba, lo que indica que la aplicación está operando normal sin alcanzar su capacidad máxima de heap asignada. Esto significa que, para esta carga de prueba con 20 usuarios concurrentes, la memoria no constituye un factor que limite el rendimiento observado, y que existe un

amplio margen antes de que la gestión de memoria se convierta en un cuello de botella.

Figura 46

Gráfica de consumo de JVM Heap Memory del aplicativo sincrónico

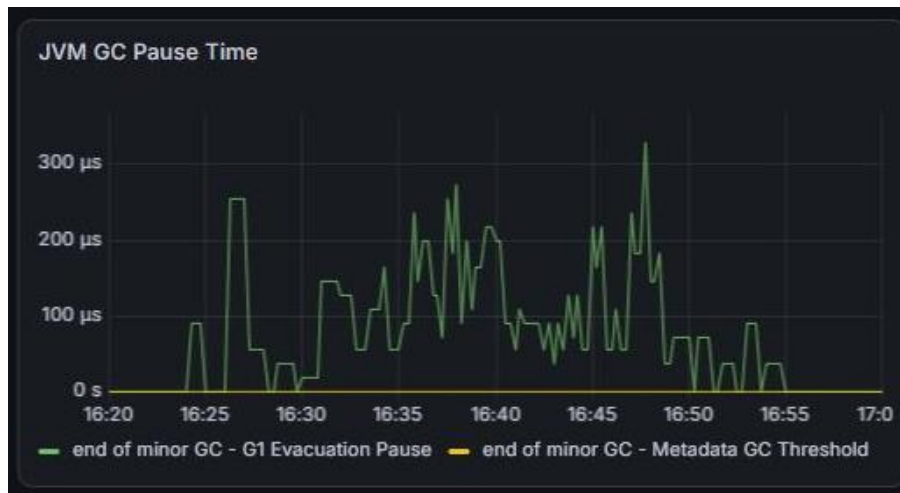


2) JVM GC Pause Time

En la siguiente figura 47 se observa el tiempo de pausa generado por el recolector de basura (Garbage Collector) durante la ejecución. Las pausas se mantienen mayormente por debajo de 250 microsegundos a lo largo de la prueba, con un pico de aproximadamente 320 microsegundos alrededor del minuto 16:48, coincidiendo con el intermedio de la prueba. El comportamiento del GC es regular y de bajo impacto, lo que indica que la gestión de memoria de la JVM no representa una fuente significativa de latencia adicional para las operaciones CRUD evaluadas.

Figura 47

Gráfica de consumo de JVM del aplicativo sincrónico



3) JVM Threads

En la siguiente figura 48 se muestra el número de hilos vivos (verde) e hilos daemon (amarillo) de la JVM. Se observa un incremento desde 20 hilos en reposo hasta estabilizarse entre 35-36 hilos activos durante la fase de carga en el tiempo (23:35-23:55), y luego un descenso a los valores base al finalizar la prueba. Este comportamiento es el esperado, donde la petición HTTP entrante ocupa un hilo dedicado durante toda su ejecución de procesamiento + espera de respuesta de la base de datos. El número de hilos supera ese valor ya que tiene valor de 35-36 esto refleja un adicional de hilos internos del framework y del contenedor de aplicaciones Tomcat y no solo los hilos dedicados directamente a las peticiones de usuario.

Figura 48

Gráfica de consumo de JVM Threads del aplicativo sincrónico



d) Conexiones HikariCP

En la siguiente figura 49 se muestra el estado del pool de conexiones a la base de datos, conexiones activas (verde), idle/libres (amarillo) y en espera/pending (azul). Se observa que, al iniciar la carga en el tiempo 16:25, las conexiones activas se incrementan y alcanzan rápidamente su máximo configurado de 15 conexiones en el tiempo de 16:36 y se mantienen en ese tope durante toda la fase, mientras que simultáneamente aparecen conexiones en espera (pending), estabilizándose en aproximadamente 5. Esto indica que el pool de conexiones se saturó bajo la carga de 20 usuarios concurrentes, y que existían peticiones adicionales esperando a que se liberó una conexión antes de poder ejecutar la siguiente operación contra la base de datos generando un cuello de botella directamente y esto es atribuible al tamaño configurado del pool y al modelo de conexión bloqueante, donde cada hilo retiene su conexión durante todo el ciclo de vida de la petición, los idle son conexiones hechas y listas para usarse es por eso en el tiempo 16:35 están libres y en ese mismo pasan a 0 y las pendientes lo toman.

Figura 49

Gráfica de conexiones de HikariCP del aplicativo sincrónico



e) Tiempo de espera de conexión

En la siguiente figura 50 se muestra el tiempo que una petición debe esperar para obtener una conexión libre del pool. Se observa un salto de 0 ms a aproximadamente 5 segundos constantes entre los minutos 16:35 y 16:45. Esto significa que, durante gran parte de la prueba, una porción significativa del tiempo total de respuesta de la aplicación no correspondía al procesamiento real de la operación CRUD, sino a la espera por obtener una conexión disponible hacia SQL Server.

Figura 50

Gráfica de tiempo de espera de conexión del aplicativo sincrónico

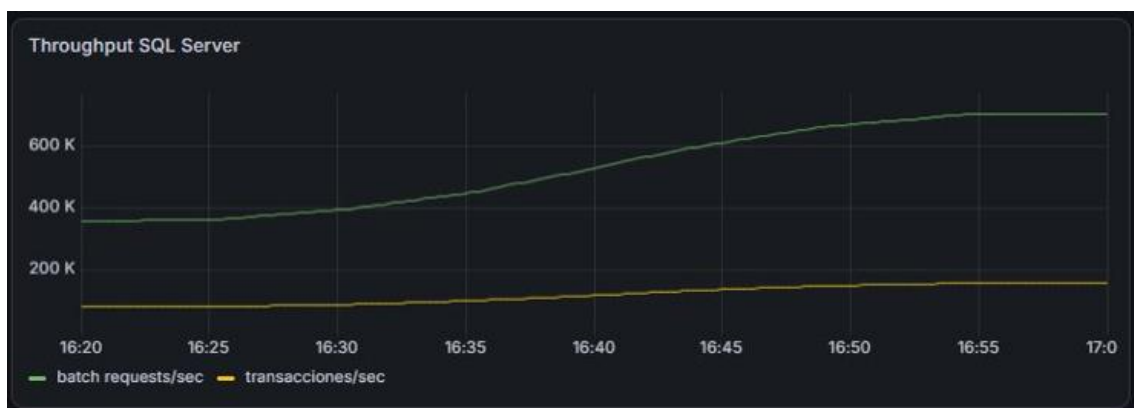


f) Throughput SQL Server

En la siguiente figura 51 se muestran dos métricas batch requests/sec (verde), que mide lotes de comandos SQL ejecutados, y transacciones/sec (amarillo). Ambas curvas muestran un crecimiento sostenido desde el inicio de la prueba hasta estabilizarse alrededor del minuto 16:45, alcanzando aproximadamente 650K batch requests acumulados y 100K transacciones acumuladas, 650k es el acumulado histórico.

Figura 51

Gráfica de Throughput del consumo de SQL server del aplicativo sincrónico



g) Locks y deadlocks

En la siguiente figura 52 se observa un crecimiento sostenido de los lock waits durante la fase de carga, alcanzando aproximadamente 12.5K esperas acumuladas antes de estabilizarse, mientras que la línea de deadlocks se mantiene prácticamente en 0 durante toda la prueba. Esto indica que, si existieron esperas por bloqueos de fila/tabla y no se

registraron deadlocks que hubieran obligado a SQL Server a abortar transacciones, lo cual es confiable con la tasa de error del 0% observada en JMeter.

Figura 52

Gráfica de lock y deadlock del aplicativo sincrónico

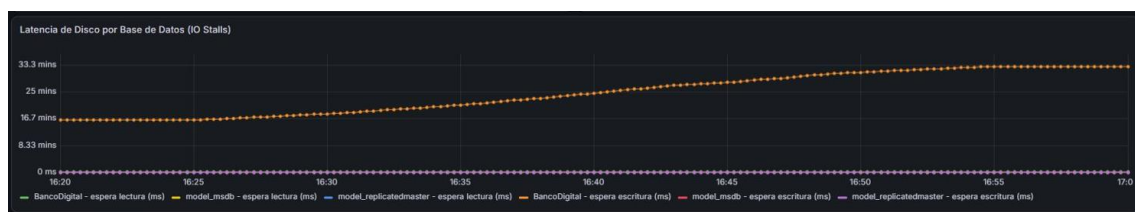


h) Latencia de Disco por base de datos

En la siguiente figura 53 se mide el tiempo acumulado de espera en operaciones de lectura/escritura a nivel de disco físico. La base de datos BancoDigital, para realizar una escritura (naranja) presenta el crecimiento más alto, acumulando hasta aproximadamente 30.0 minutos de espera total de I/O de escritura hacia el final de la prueba, mientras que las esperas de lectura y las de las bases de datos del sistema (model_msdb, model_replicatedmaster) permanecen cercanas a 0. Esto significa que las operaciones CRUD con proporción significativa de escrituras: create, update, delete, lo cual es coherente la mayor parte de la latencia a nivel de disco proviene específicamente de las operaciones de escritura sobre la base de datos de la aplicación, no de operaciones de lectura ni de procesos internos del motor de base de datos.

Figura 53

Gráfica de latencia del disco del aplicativo sincrónico



4.5.2.1.5 Estadística descriptiva de la prueba de estrés del aplicativo sincrónico

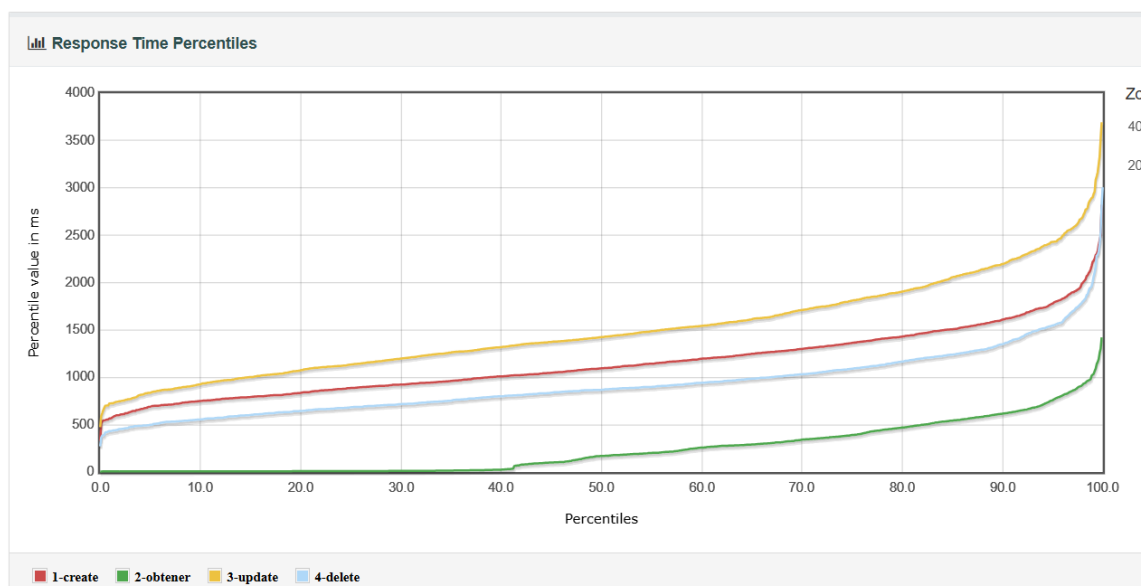
En la siguiente figura 54 se muestra el tiempo de respuesta acumulado por percentil para cada operación CRUD:

- a) **Create (rojo):** muestra una mediana de aproximadamente 1150 ms y un percentil 90 de aproximadamente 1650 ms.
- b) **Read(verde):** mantiene los valores más bajos en todo el rango, con un percentil 50 (mediana) de aproximadamente 250-300 ms y un percentil 90 cercano a 550-600 ms.
- c) **Update (amarillo):** presenta los valores más altos en todos los percentiles, con una mediana de aproximadamente 1750 ms y un percentil 90 cercano a 2300 ms.
- d) **Delete (celeste):** presenta una mediana de aproximadamente 900 ms y un percentil 90 cercano a 1400 ms.

En el extremo superior percentil 99-100, las cuatro curvas muestran un incremento, alcanzando valores máximos de aproximadamente 3400 ms para update, 2750 ms para create, 2050 ms para delete y 2000 ms para obtener, lo que evidencia que, existe un subconjunto de solicitudes con tiempos de respuesta considerablemente más altos que el resto.

Figura 54

Gráfica de percentiles del aplicativo sincrónico



Validación de los resultados

a) Tasa de error

Se registró una tasa de error del 0.00% en las cuatro operaciones evaluadas sobre un total de 26 425 mediciones, lo que indica que el aplicativo no presentó fallos funcionales durante toda la prueba, sin fallos, excepciones ni interrupciones de conexión registradas en ninguna de las operaciones.

- a) **Create:** 4989 ejecuciones, 0 fallos (0.00%)
- b) **Read:** 4970 ejecuciones, 0 fallos (0.00%)
- c) **Update:** 4969 ejecuciones, 0 fallos (0.00%)
- d) **Delete:** 4950 ejecuciones, 0 fallos (0.00%)

b) Error estándar de la media

El error estándar de la media ($SE = \text{desviación estándar} / \sqrt{n}$) resulta pequeño en relación con el promedio de cada operación, lo que indica que los valores constituyen estimaciones precisas y confiables del verdadero comportamiento del sistema, esto mide cuando variaría el promedio si repitiera nuevamente la prueba.

- a) **Create:**
 - a. $SE = 349.74 / \sqrt{4989} = 4.95$ ms sobre un promedio de 1147 ms = 0.43%
- b) **Read:**
 - a. $SE = 265.72 / \sqrt{4970} = 3.21$ ms sobre un promedio de 238 ms = 1.5%
- c) **Update:**
 - a. $SE = 499.31 / \sqrt{4969} = 5.83$ ms sobre un promedio de 1507 ms = 0.47%
- d) **Delete:**
 - a. $SE = 336.53 / \sqrt{4950} = 4.12$ ms sobre un promedio de 924 ms = 0.51%

c) Cercanía entre promedio y mediana

Se evidencia una cercanía similar entre el promedio y la mediana en las cuatro operaciones evaluadas, con diferencias mínimas en todos los casos, lo que confirma que los resultados no se encuentran distorsionados por valores extremos y que la distribución de tiempos de respuesta es relativamente simétrica.

- a) **Create:** promedio 1147 ms frente a mediana 1095 ms (diferencia de 4.5%)
- b) **Read:** promedio 238 ms frente a mediana 169 ms (diferencia de 28.9%)
- c) **Update:** promedio 1507 ms frente a mediana 1425 ms (diferencia de 5.4%)
- d) **Delete:** promedio 924 ms frente a mediana 868 ms (diferencia de 6.06%)

d) Progresión gradual de percentiles

En las cuatro operaciones se observa un incremento progresivo y sin saltos abruptos entre los percentiles 50, 90, 95 y 99, lo que descarta la presencia de comportamientos erráticos o bimodales en el sistema durante la prueba.

- a) **Create:** mediana 1095 ms, → 90% 1610 ms, → 95% 1778 ms, → 99% 2220 ms
- b) **Read:** mediana 169 ms, → 90% 615 ms, → 95% 758 ms, → 99% 1028 ms
- c) **Update:** mediana 142 ms, → 90% 2197 ms, → 95% 2431 ms, → 99% 2902 ms
- d) **Delete:** mediana 868 ms, → 90% 1350 ms, → 95% 1547 ms, → 99% 2009 ms

e) Throughput similar entre operaciones

El throughput se mantuvo prácticamente idéntico entre las cuatro operaciones, lo que evidencia un procesamiento predecible y balanceado del sistema bajo la carga aplicada, independientemente del tiempo de respuesta individual de cada operación.

- a) **Create:** 2.8 transacciones/seg
- b) **Read:** 2.8 transacciones/seg
- c) **Update:** 2.8 transacciones/seg
- d) **Delete:** 2.8 transacciones/seg

f) Volumen y duración suficientes para diluir el ruido de corto plazo

El volumen total de mediciones recolectadas durante los 30 minutos de prueba continua resulta suficiente para que efectos puntuales de corta duración.

- a) **Create:** 4989 mediciones
- b) **Read:** 4970 mediciones
- c) **Update:** 4969 mediciones
- d) **Delete:** 4950 mediciones

Total: 19 878 mediciones en 30 minutos de prueba continua

4.5.2.2 Resultados del aplicativo asincrónico

Los resultados que se tuvo para el aplicativo asincrónico fueron los siguientes:

Tabla 10

Resultados del aplicativo asincrónico en la prueba de estrés

Endpoint	Cantidad de ejecuciones	Tiempo de respuesta promedio (ms)	Tiempo mínimo de respuesta (ms)	Tiempo máximo de respuesta (ms)	Desviación estándar	Throughput	Mediana	90%	95%
Creación	6189	1077	310	5544	297.82	3.5/sec.	1041	1475	1606
Obtener	6167	265	8	2296	165.42	3.5/sec.	257	452	557
Edición	6164	842	167	4327	379.03	3.5/sec.	760	1381	1559
Eliminación	6152	892	240	4321	311.36	3.5/sec.	834	1316	1484
Total	24 672	769	8	5544	426.18	14.0/sec.	765	1322	1499

Nota. Elaboración propia.

Se detalla a continuación sobre cada endpoint sus métricas:

- a) **Resultado del endpoint Creación:** Se ejecutó 6189 veces, con un promedio de respuesta de 1077 ms, el mínimo tiempo de respuesta fue de 310 ms y el máximo de 5544 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 297.82, con un throughput de 3.5/sec., con una mediana de 1041 y percentil 90 de 1475 y percentil 95 de 1606 en JMeter.
- b) **Resultado del endpoint Obtener:** Se ejecutó 6167 veces, con un promedio de respuesta de 265 ms, el mínimo tiempo de respuesta fue de 8 ms y el máximo de 2296 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 165.42, con un throughput de 3.5/sec., con una mediana de 257 y percentil 90 de 452 y percentil 95 de 557 en JMeter.
- c) **Resultado del endpoint Edición:** Se ejecutó 6164 veces, con un promedio de respuesta de 842 ms, el mínimo tiempo de respuesta fue de 167 ms y el máximo de 4327 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 379.03, con un throughput de 3.5/sec., con una mediana de 760 y percentil 90 de 1381 y percentil 95 de 1559 en JMeter.

d) Resultado del endpoint Eliminación: Se ejecutó 6152 veces, con un promedio de respuesta de 892 ms, el mínimo tiempo de respuesta fue de 240 ms y el máximo de 4321 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 311.36, con un throughput de 3.5/sec., con una mediana de 834 y percentil 90 de 1316 y percentil 95 de 1484 en JMeter.

4.5.2.2.1 Latencia

a) Latencia durante la prueba

En la siguiente figura 55 se muestra la latencia promedio en el tiempo transcurrido de la prueba para las 4 operaciones CRUD. El gráfico muestra latencia promedio vs el tiempo de la prueba, en la cual mide específicamente el tiempo hasta recibir el primer byte de respuesta del servidor.

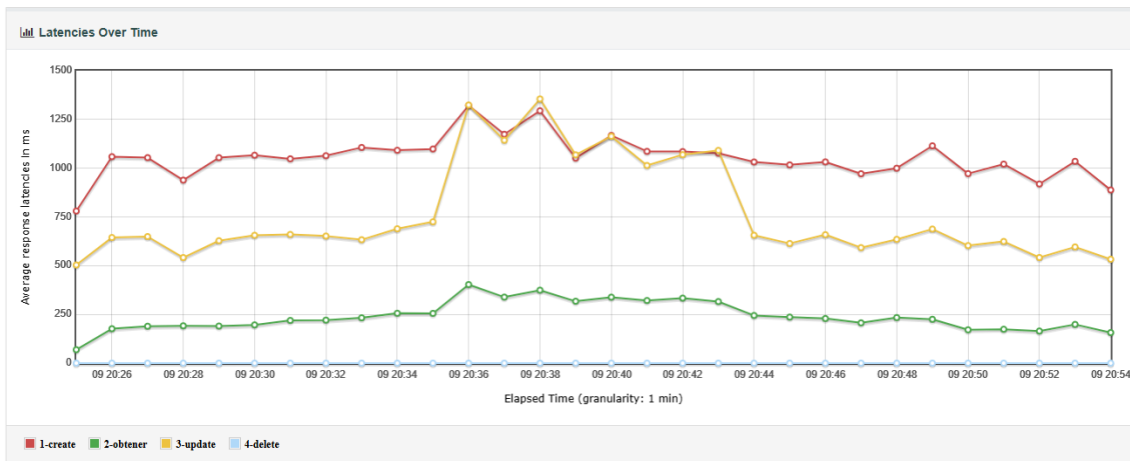
Se observa un comportamiento similar para cada operación donde hay una fase de ascenso esto sería normal ya que en los primeros minutos de la prueba específicamente 10 minutos va creciendo los usuarios hasta llegar a los 20 usuarios, seguida de una fase de estable de 8 minutos y un descenso en los últimos 12 minutos ya que los usuarios están disminuyendo para terminar la prueba.

Por operación se detalla lo siguiente:

- a) **Create (rojo):** Se ubica en un rango intermedio, estabilizándose entre 1000 a 1250ms con un pico máximo de 1295 en el minuto 20:38.
- b) **Read (verde):** Se muestra una latencia considerablemente menor con respecto a los endpoint de create y update, estabilizándose entre 250 a 300ms, esto normal ya que las operaciones de lectura son más rápidas.
- c) **Update (amarillo):** Presenta una media, estabilizándose entre 600 - 1250ms, con un pico de máximo de 1355 en el minuto 20:38.
- d) **Delete (azul):** Se mantiene en 0ms durante la prueba ya que para esta operación no se espera ni una respuesta(response) que tenga contenido, es por eso que la respuesta no tiene un byte para poder calcularlo.

Figura 55

Gráfica de latencia del aplicativo asincrónico



4.5.2.2.2 Response Time

a) Tiempo de respuesta durante la prueba

En el siguiente gráfico 56 se muestra el tiempo de respuesta promedio en el tiempo que duro la prueba, se puede observar el grafico de las 4 operaciones (CRUD). El tiempo de respuesta mide el proceso completo de la solicitud, desde el envío hasta la recepción total de la respuesta.

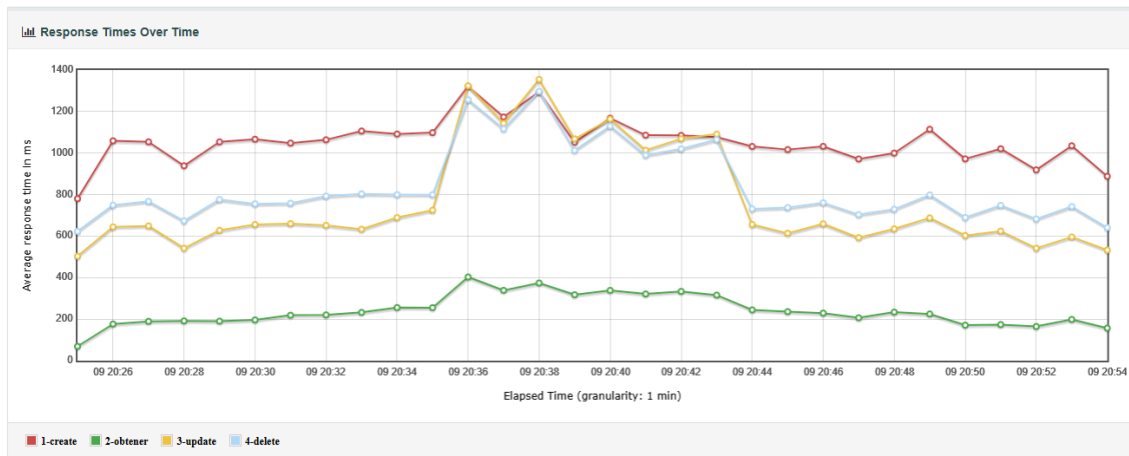
En el gráfico se identifican también 3 fases que la primera es la de creciente o ascenso en la cual se puede observar que en los primeros minutos hasta el minuto 12 se muestra una elevación ya que en esos momentos se están incrementando los usuarios hasta llegar hasta los 20 usuarios, después se mantiene una estabilidad durante los 8 minutos ya que se encuentra activos los 20 usuarios, después una fase de descenso en la cual los usuarios van decreciendo para terminar la prueba. Se detalla por operación:

- Create (rojo):** Se ubica en un rango de 1000 a 1250, con el punto más alto en 1295 en el tiempo 20:38. Estos comportamientos es propio de una operación de escritura que requiere insertar registros a las tablas de cuenta, saldo y operación en todo el proceso.
- Read (verde):** Es la operación más rápida, ya que oscila entre 250 a 300 durante toda la fase. Esto es rápido ya que no bloquea las tablas al momento de consultar. Teniendo un punto máximo de 403ms en el tiempo 20:36.
- Update (amarillo):** Presentan el tiempo de respuesta más alto, estabilizándose entre 600 y 1250ms, con su pico máximo de 1355ms en el tiempo 20:38 de la prueba, esto debido que busca los registros en cada tabla, realiza la modificación y guardado del registro en las tablas teniendo una mayor sobrecarga transaccional.

- d) **Delete(azul):** Es el proceso que se ubica en una estabilización de 700 a 1355, un valor estable ya que se encuentra en tiempos menores de create y update ya que este proceso se encarga de buscar y eliminar el registro, teniendo un tiempo de respuesta más alto en 1297 en el tiempo 20:36.

Figura 56

Gráfica de tiempo de respuesta durante la prueba



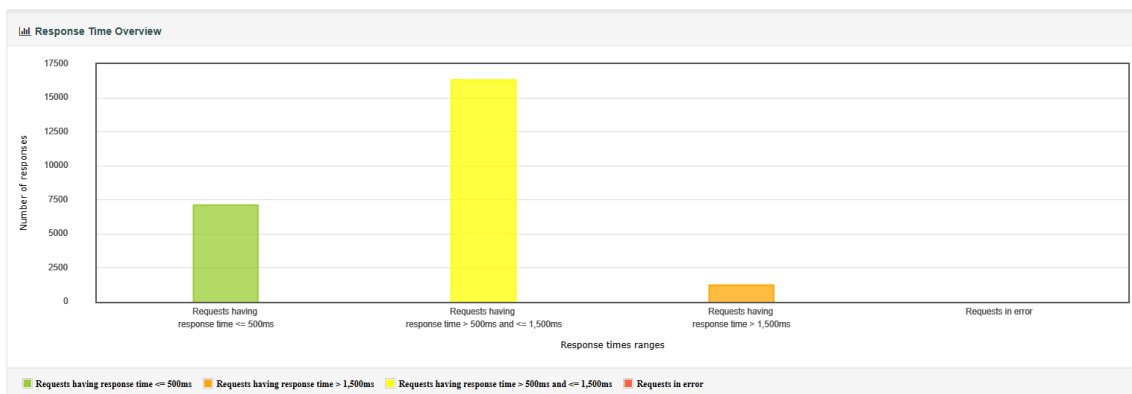
b) Distribución de cantidad de peticiones y tiempo de respuesta

En la siguiente figura 57 se muestra la distribución general de cantidades de respuestas, agrupando según el rango de tiempo que duro la peticiones se dividen en 3 partes:

- Tiempo de respuesta de ≤ 500 ms: Se 7120 peticiones
- Tiempo de respuesta entre $500 < > 1500$: 16328 peticiones
- Tiempo de respuesta mayor de >1500 : 1224 peticiones

Figura 57

Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo asincrónico



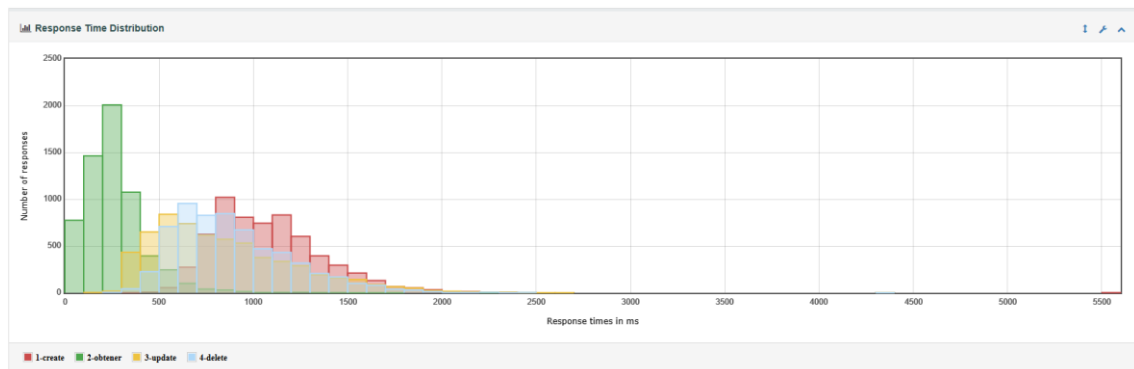
c) Distribución de tiempo de respuesta

En la siguiente figura 58 se muestra la distribución de frecuencias de tiempo de respuesta, separada por operación CRUD.

- a) **Create (rojo):** Se distribuye en mayor concentración en 800 a 1200ms, teniendo la mayor distribución en el rango de 1020 en 800 a 900ms.
- b) **Read (verde):** Varias peticiones en mayor se concentran en el rango de 0 a 400ms, teniendo en la mayor distribución de 2007 peticiones en 200 a 300 ms.
- c) **Update (amarillo):** Se distribuye en mayor concentración en 300 a 1000ms, teniendo la mayor distribución con 839 peticiones en 500 a 600ms.
- d) **Delete(azul):** Se distribuye principalmente en 500 a 1000ms, teniendo en la mayoría 954 peticiones en 600 a 700ms.

Figura 58

Gráfica de distribución de tiempo de respuesta



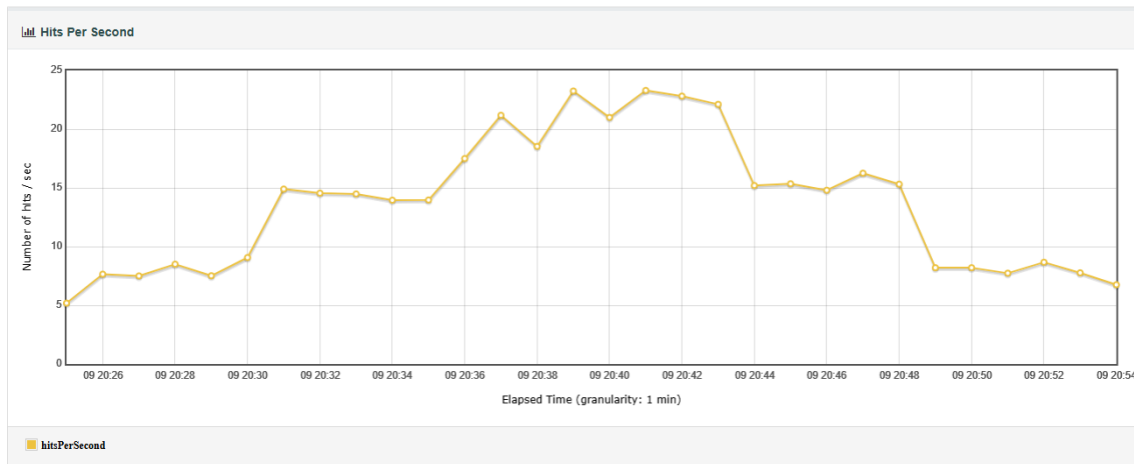
4.5.2.2.3 Throughput

a) TPS en general

En la siguiente figura 59 se puede observar el total de los TPS en el transcurso de la prueba como va variando en todo el transcurso, en este caso es la suma general de las 4 operaciones CRUD, se puede observar que hay un ascenso cuando empieza en los primeros minutos es normal ya que varios usuarios están siendo activados para realizar peticiones y dentro de los 8 minutos se mantiene estable manteniendo un rango de 15 a 24 TPS y teniendo un pico máximo de 23:25 en el tiempo 20:39 y en los últimos 5 minutos teniendo un descenso ya que usuarios se van disminuyendo.

Figura 59

Gráfica de TPS en general del aplicativo asincrónico



b) TPS por operación

En la siguiente figura 60 se muestra el throughput (transacciones por segundo) de las cuatro operaciones CRUD a lo largo del tiempo de prueba. Acá se observa que las 4 operaciones se comportan de manera prácticamente idénticas.

- a) **Create (azul):** Se puede observar que alcanza su punto máximo de 5.9 en el tiempo de 20:41.
- b) **Read (amarillo):** Se puede observar que alcanza su punto máximo de 5.9 en el tiempo de 20:41.
- c) **Update (rojo):** Se puede observar que alcanza su punto máximo de 5.9 en el tiempo de 20:41.
- d) **Delete(verde):** Se puede observar que alcanza su punto máximo de 5.9 en el tiempo de 20:41.

Figura 60

Gráfica del TPS por operación del aplicativo asincrónico



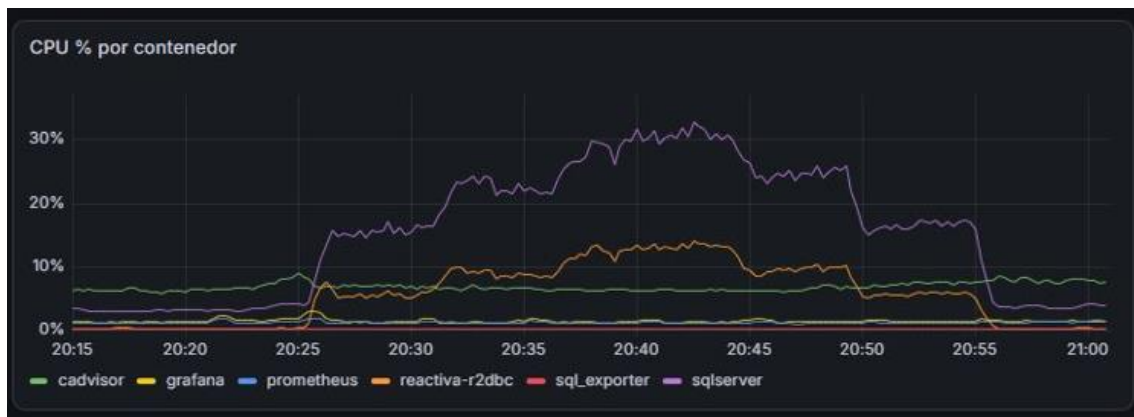
4.5.2.2.4 Consumo de recursos

a) Consumo de CPU

En la siguiente figura 61 se observa el consumo de CPU por contenedor Docker durante los 30 minutos de prueba. El contenedor de SQL Server (morado) presenta el consumo más alto, alcanzando picos de aproximadamente 30-32% durante la fase estable (20:40-20:45), mientras que el contenedor de la aplicación reactivo (naranja) se mantiene en un rango de 8-15%, con un pico de 15% en el intermedio de la prueba (23:20). Los contenedores de monitoreo (cAdvisor, Grafana, Prometheus) presentan un consumo pequeño, por debajo del 10%, confirmando que no interfieren en la medición.

Figura 61

Gráfica del consumo de CPU del aplicativo asincrónico

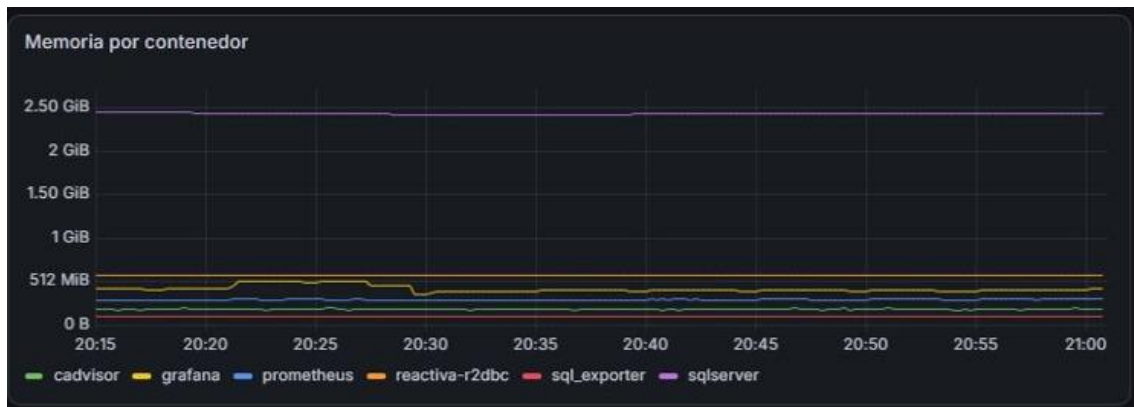


b) Consumo de memoria

En la siguiente figura 62 se muestra la memoria se mantiene prácticamente constante durante toda la prueba, sin picos ni variaciones asociadas al aumento de carga. El contenedor de SQL Server (morado) consume de forma estable 2.5 GiB, mientras que la aplicación reactiva (naranja) se mantiene alrededor de 512 MiB.

Figura 62

Gráfica de consumo de memoria del aplicativo asincrónico



c) Consumo de JVM

1) JVM Heap Memory

En la siguiente figura 63 se muestra el uso del heap de la JVM de manera desagregado por generación: G1 Eden Space, G1 Old Gen y G1 Survivor Space, junto con el límite máximo configurado. Los tres componentes se mantienen en valores estables durante toda la prueba, lo que indica que la aplicación está operando normal sin alcanzar su capacidad máxima de heap asignada. Esto significa que, para esta carga de prueba con 20 usuarios concurrentes, la memoria no constituye un factor que limite el rendimiento observado, y que existe un amplio margen antes de que la gestión de memoria se convierta en un cuello de botella.

Figura 63

Gráfica de consumo de JVM Heap Memory del aplicativo asincrónico



2) JVM GC Pause Time

En la siguiente figura 64 se observa el tiempo de pausa generado por el recolector de basura (Garbage Collector) durante la ejecución. Las pausas se mantienen mayormente por debajo de 300 microsegundos a lo largo de la prueba, con un pico de aproximadamente 320 microsegundo alrededor del minuto 20:42, coincidiendo con el inicio de la prueba. El comportamiento del GC es regular y de bajo impacto, lo que indica que la gestión de memoria de la JVM no representa una fuente significativa de latencia adicional para las operaciones CRUD evaluadas.

Figura 64

Gráfica de consumo de JVM GC Pause Time del aplicativo asincrónico



3) JVM Threads

En la siguiente figura 65 se muestra el número de hilos vivos (verde) e hilos daemon (amarillo) de la JVM. Se observa un incremento desde 20 hilos en reposo hasta estabilizarse entre 28 a 32 hilos activos durante la fase de carga, y luego un descenso a los valores base al finalizar la prueba. Este comportamiento es el esperado, donde la petición HTTP entrante ocupa un hilo dedicado durante toda su ejecución de procesamiento + espera de respuesta de la base de datos. El número de hilos supera ese valor ya que tiene valor de 28 a 32 esto refleja un adicional de hilos internos del framework y del contenedor de aplicaciones Tomcat y no solo los hilos dedicados directamente a las peticiones de usuario.

Figura 65

Gráfica de consumo de JVM Threads del aplicativo asincrónico



d) Conexiones R2DBC

En la siguiente figura 66 se muestra el estado del pool de conexiones a la base de datos, conexiones activas (verde), idle/libres (amarillo) y en espera/pending (azul). Se observa que, al iniciar la carga en el tiempo 20:25, las conexiones activas alcanzan rápidamente su máximo configurado de 15 conexiones en los primeros 10 minutos y se mantienen en ese tope durante toda la fase, mientras que simultáneamente aparecen conexiones en espera (pending), teniendo mucha variabilidad. Esto indica que el pool de conexiones se saturó bajo la carga de 20 usuarios concurrentes, y que existían peticiones adicionales esperando a que se liberé una conexión antes de poder ejecutar la siguiente operación contra la base de datos generando un cuello de botella directamente y esto es un atribuible al tamaño configurado del pool.

Figura 66

Gráfica de conexiones de R2dbc del aplicativo asincrónico



e) Throughput SQL Server

En la siguiente figura 67 se muestran dos métricas batch requests/sec (verde), que mide lotes de comandos SQL ejecutados, y transacciones/sec (amarillo). Ambas curvas muestran un crecimiento sostenido desde el inicio de la prueba hasta estabilizarse alrededor del minuto 21:00, alcanzando aproximadamente 1.25M batch requests acumulados y 100K transacciones acumuladas.

Figura 67

Gráfica de Throughput del consumo de SQL server del aplicativo asincrónico



f) Locks y deadlocks

En la siguiente figura 68 se observa un crecimiento sostenido de los lock waits durante la fase de carga, alcanzando aproximadamente 20K esperas acumuladas antes de estabilizarse, mientras que la línea de deadlocks se mantiene prácticamente en 0 durante toda la prueba. Esto indica que, si existieron esperas por bloqueos de fila/tabla y no se registraron deadlocks que hubieran obligado a SQL Server a abortar transacciones, lo cual es confiable con la tasa de error del 0% observada en JMeter.

Figura 68

Gráfica de lock y deadlock del aplicativo asincrónico

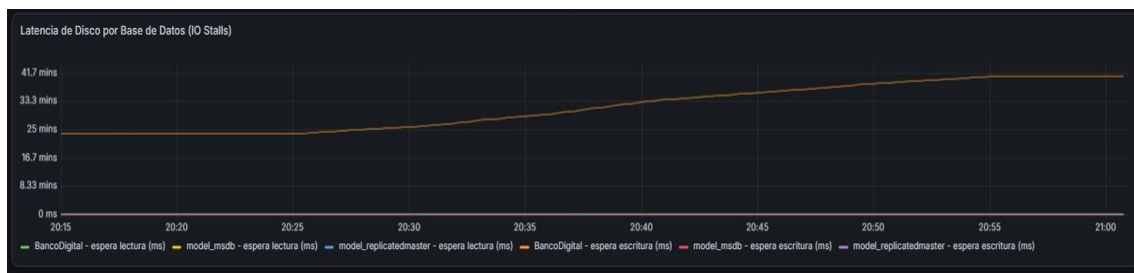


g) Latencia de Disco por base de datos

En la siguiente figura 69 se mide el tiempo acumulado de espera en operaciones de lectura/escritura a nivel de disco físico. La base de datos BancoDigital, para realizar una escritura (naranja) presenta el crecimiento más alto, acumulando hasta aproximadamente 33.3 minutos de espera total de I/O de escritura hacia el final de la prueba, mientras que las esperas de lectura y las de las bases de datos del sistema (model_msdb, model_replicatedmaster) permanecen cercanas a 0. Esto significa que las operaciones CRUD con proporción significativa de escrituras: create, update, delete, lo cual es coherente la mayor parte de la latencia a nivel de disco proviene específicamente de las operaciones de escritura sobre la base de datos de la aplicación, no de operaciones de lectura ni de procesos internos del motor de base de datos.

Figura 69

Gráfica de latencia del disco del aplicativo asincrónico



4.5.2.2.5 Estadística descriptiva de la prueba de estrés del aplicativo asincrónico

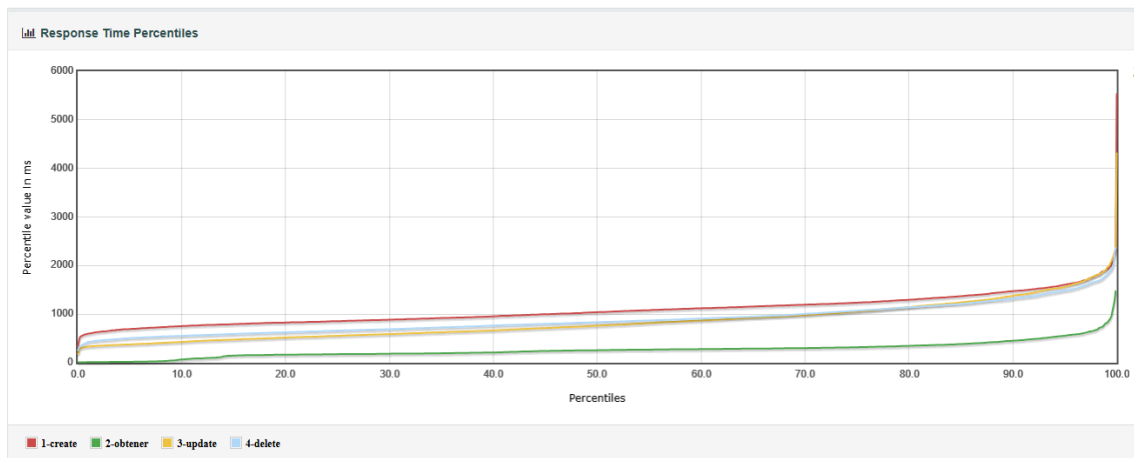
En la siguiente figura 70 se muestra el tiempo de respuesta acumulado por percentil para cada operación CRUD:

- Create (rojo):** muestra una mediana de aproximadamente 1040 ms y un percentil 90 de aproximadamente 1475 ms.
- Read(verde):** mantiene los valores más bajos en todo el rango, con un percentil 50 (mediana) de aproximadamente 257 ms y un percentil 90 cercano a 451 ms.
- Update (amarillo):** presenta los valores más altos en todos los percentiles, con una mediana de aproximadamente 758 ms y un percentil 90 cercano a 1384 ms.
- Delete (celeste):** presenta una mediana de aproximadamente 836 ms y un percentil 90 cercano a 1315 ms.

En el extremo superior percentil 99-100, las cuatro curvas muestran un incremento, alcanzando valores máximos de aproximadamente 1923 ms para update, 1923 ms para create, 1788 ms para delete y 814 ms para obtener, lo que evidencia que, existe un subconjunto de solicitudes con tiempos de respuesta considerablemente más altos que el resto.

Figura 70

Gráfica de percentiles del aplicativo asincrónico



Validación de los resultados

a) Tasa de error

Se registró una tasa de error del 0.00% en las cuatro operaciones evaluadas sobre un total de 26 425 mediciones, lo que indica que el aplicativo no presentó fallos funcionales durante toda la prueba, sin fallos, excepciones ni interrupciones de conexión registradas en ninguna de las operaciones.

- a) **Create:** 6189 ejecuciones, 0 fallos (0.00%)
- b) **Read:** 6167 ejecuciones, 0 fallos (0.00%)
- c) **Update:** 6164 ejecuciones, 0 fallos (0.00%)
- d) **Delete:** 6152 ejecuciones, 0 fallos (0.00%)

b) Error estándar de la media

El error estándar de la media ($SE = \text{desviación estándar} / \sqrt{n}$) resulta pequeño en relación con el promedio de cada operación, lo que indica que los valores constituyen estimaciones precisas y confiables del verdadero comportamiento del sistema, esto mide cuando variaría el promedio si repitiera nuevamente la prueba.

a) **Create:**

a. $SE = 297.82 / \sqrt{6189} = 3.78$ ms sobre un promedio de 1077 ms = 0.35%

b) **Read:**

a. $SE = 165.42 / \sqrt{6167} = 2.10$ ms sobre un promedio de 265 ms = 0.79%

c) **Update:**

a. $SE = 379.03 / \sqrt{6164} = 4.82$ ms sobre un promedio de 842 ms = 0.57%

d) **Delete:**

a. $SE = 311.36 / \sqrt{6152} = 3.96$ ms sobre un promedio de 892 ms = 0.44%

c) Cercanía entre promedio y mediana

Se evidencia una cercanía similar entre el promedio y la mediana en las cuatro operaciones evaluadas, con diferencias mínimas en todos los casos, lo que confirma que los resultados no se encuentran distorsionados por valores extremos y que la distribución de tiempos de respuesta es relativamente simétrica.

a) **Create:** promedio 1077 ms frente a mediana 1041 ms (diferencia de 3.3%)

b) **Read:** promedio 265 ms frente a mediana 257 ms (diferencia de 3.01%)

c) **Update:** promedio 842 ms frente a mediana 760 ms (diferencia de 9.7%)

d) **Delete:** promedio 892 ms frente a mediana 834 ms (diferencia de 6.5%)

d) Progresión gradual de percentiles

En las cuatro operaciones se observa un incremento progresivo y sin saltos abruptos entre los percentiles 50, 90, 95 y 99, lo que descarta la presencia de comportamientos erráticos o bimodales en el sistema durante la prueba.

a) **Create:** mediana 1041 ms, → 90% 1475 ms, → 95% 1606 ms, → 99% 1915 ms

b) **Read:** mediana 257 ms, → 90% 452 ms, → 95% 557 ms, → 99% 814 ms

c) **Update:** mediana 760 ms, → 90% 1381 ms, → 95% 1559 ms, → 99% 1934 ms

d) **Delete:** mediana 834 ms, → 90% 1316 ms, → 95% 1484 ms, → 99% 1798 ms

e) Throughput similar entre operaciones

El throughput se mantuvo prácticamente idéntico entre las cuatro operaciones, lo que evidencia un procesamiento predecible y balanceado del sistema bajo la carga aplicada, independientemente del tiempo de respuesta individual de cada operación.

a) **Create:** 3.50 transacciones/seg

- b) **Read:** 3.5transacciones/seg
- c) **Update:** 3.5 transacciones/seg
- d) **Delete:** 3.5 transacciones/seg

f) Volumen y duración suficientes para diluir el ruido de corto plazo

El volumen total de mediciones recolectadas durante los 30 minutos de prueba continua resulta suficiente para que efectos puntuales de corta duración.

- a) **Create:** 6189 mediciones
- b) **Read:** 6167 mediciones
- c) **Update:** 6164 mediciones
- d) **Delete:** 6152 mediciones

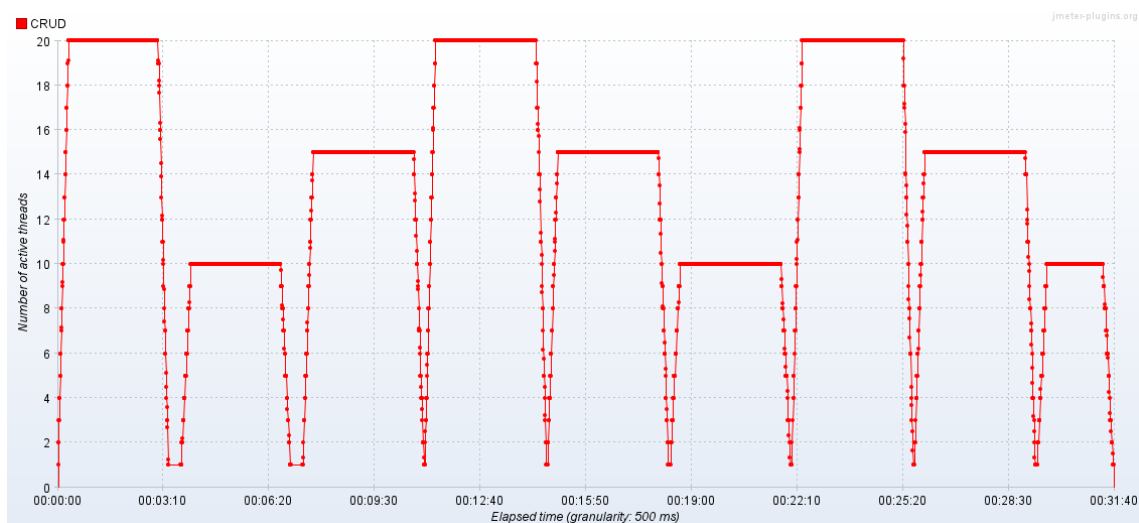
Total: 26 425 mediciones en 30 minutos de prueba continua

4.5.3 Pruebas de picos

Como se muestra la figura 71, para esta prueba de picos se ejecutaron 60 hilos (usuarios virtuales) con una duración de 30 minutos, de los cuales se ejecutaron 20,10,15,20,15,10,20,15,10 en varios tiempos ejecutando por 3 minutos constante después llegando a 0 usuarios y después los siguientes usuarios simulando una alza y caída de usuarios.

Figura 71

Prueba de picos



4.5.3.1 Resultados del aplicativo sincrónico

Los resultados que se tuvo para el aplicativo sincrónico fueron los siguientes:

Tabla 11

Resultados del aplicativo sincrónico en la prueba de picos

Endpoint	Cantida d de ejecucio nes	Tiemp o de respues ta prome dio (ms)	Tiem po de míni mo de respu esta (ms)	Tiemp o de máxim o de respu esta (ms)	Desviaci ón estándar	Through put	Medi ana	90%.	95%
Creación	6648	1137	322	3161	340.55	3.5/sec.	1086	1602	1797
Obtener	6606	259	5	1762	234.89	3.5/sec.	223	569	705
Edición	6597	1555	472	3904	477.46	3.5/sec.	1472	2221	2460
Eliminación	6548	933	231	2746	336.13	3.5/sec.	868	1385	1572
Total	26399	971	5	3904	589.56	13.9/sec.	953	1729	1987

Nota. Elaboración propia.

Se detalla a continuación sobre cada endpoint sus métricas:

- Resultado del endpoint Creación:** Se ejecutó 6648 veces, con un promedio de respuesta de 1137 ms, el mínimo tiempo de respuesta fue de 322 ms y el máximo de 3161 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 340.55, con un throughput de 3.5/sec., con una mediana de 1086 y percentil 90 de 1602 y percentil 95 de 1797 en JMeter.
- Resultado del endpoint Obtener:** Se ejecutó 6606 veces, con un promedio de respuesta de 259 ms, el mínimo tiempo de respuesta fue de 5 ms y el máximo de 1762 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 234.89, con un throughput de 3.5/sec., con una mediana de 223 y percentil 90 de 569 y percentil 95 de 705 en JMeter.
- Resultado del endpoint Edición:** Se ejecutó 6597 veces, con un promedio de respuesta de 1555 ms, el mínimo tiempo de respuesta fue de 472 ms y el máximo

de 3904 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 477.46, con un throughput de 3.5/sec., con una mediana de 1472 y percentil 90 de 2221 y percentil 95 de 2460 en JMeter.

- d) **Resultado del endpoint Eliminación:** Se ejecutó 6548 veces, con un promedio de respuesta 933 ms, el mínimo tiempo de respuesta fue de 231 ms y el máximo de 2746 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 336.13, con un throughput de 3.5/sec., con una mediana de 868 y percentil 90 de 1385 y percentil 95 de 1572 en JMeter.

4.5.3.1.1 Latencia

a) Latencia durante la prueba

En la siguiente figura 72 se muestra la latencia promedio en el tiempo transcurrido de la prueba para las 4 operaciones CRUD. El gráfico muestra latencia promedio vs el tiempo de la prueba, en la cual mide específicamente el tiempo hasta recibir el primer byte de respuesta del servidor.

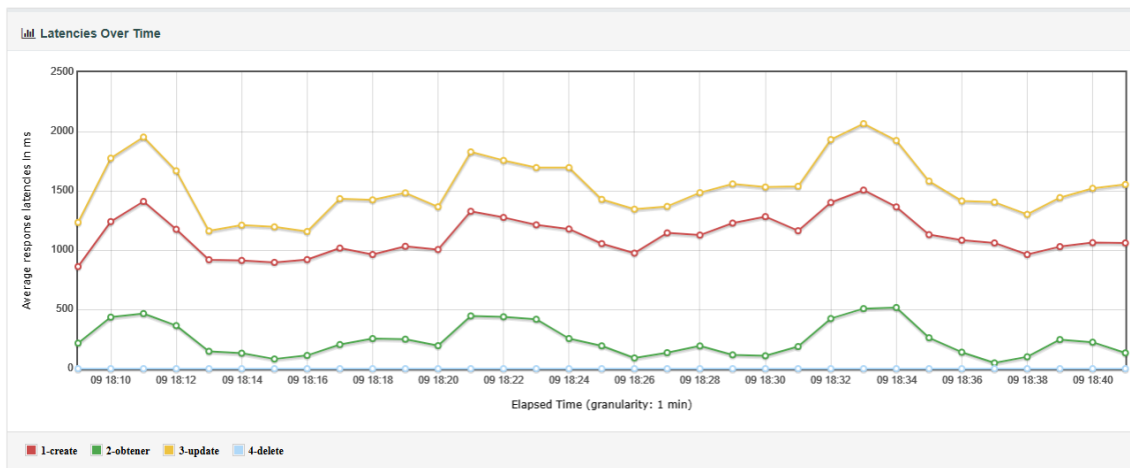
Se observa un comportamiento similar para cada operación donde hay una fase de ascenso esto sería normal ya que hay varios picos de subida donde va creciendo los usuarios hasta llegar a los 20 usuarios, seguida de una fase de estable de 2 minutos y un descenso, y después una subida así estando hasta terminar la prueba

Por operación se detalla lo siguiente:

- a) **Create (rojo):** Se ubica en un rango intermedio, estabilizándose entre 1400 a 1800ms con un pico máximo de 2067 en el minuto 18:33.
- b) **Read (verde):** Se muestra una latencia considerablemente menor con respecto a los endpoint de create y update, estabilizándose entre 400 a 500ms, esto normal ya que las operaciones de lectura son más rápidas.
- c) **Update (amarillo):** Presenta una latencia más alta de las 4, estabilizándose entre 1500 - 1800ms, con un pico de máximo de 2067 en el minuto 18:33.
- d) **Delete (azul):** Se mantiene en 0ms durante la prueba ya que para esta operación no se espera ni una respuesta(response) que tenga contenido, es por eso que la respuesta no tiene un byte para poder calcularlo.

Figura 72

Gráfica de latencia del aplicativo sincrónico



4.5.3.1.2 Response Time

a) Tiempo de respuesta durante la prueba

En el siguiente gráfico 73 se muestra el tiempo de respuesta promedio en el tiempo que duro la prueba, se puede observar el grafico de las 4 operaciones (CRUD). El tiempo de respuesta mide el proceso completo de la solicitud, desde el envío hasta la recepción total de la respuesta.

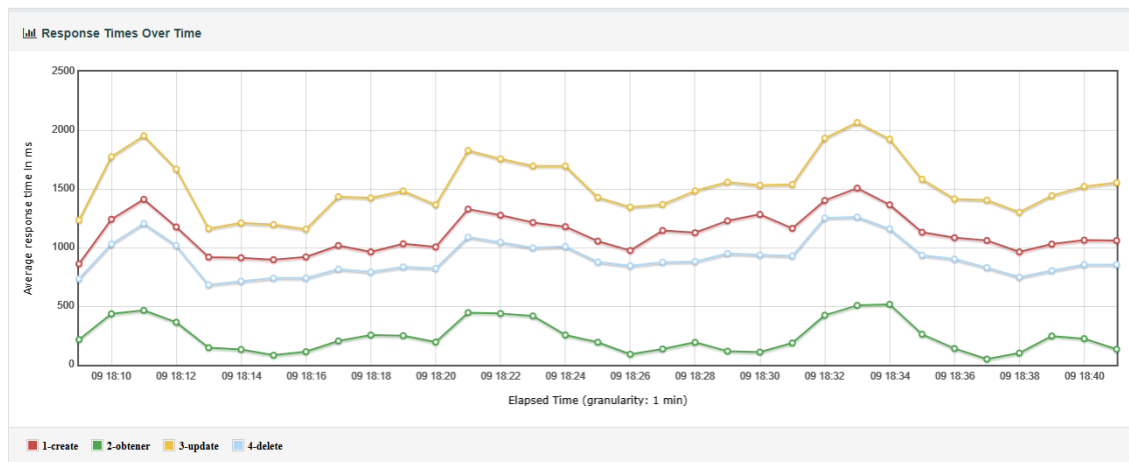
En el gráfico se identifican también 3 fases que la primera es la de creciente o ascenso en la cual se puede observar que en los primeros minutos hasta el minuto 5 se muestra una elevación ya que en esos momentos se están incrementando los usuarios hasta llegar hasta los 20 usuarios, después se mantiene una estabilidad durante los 20 minutos ya que se encuentra activos los 20 usuarios, después una fase de descenso en la cual los usuarios van decreciendo para terminar la prueba. Se detalla por operación:

- Create (rojo):** Se ubica en un rango de 1400 a 1800, con el punto más alto en 2067 en el tiempo 18:33. Estos comportamientos es propio de una operación de escritura que requiere insertar registros a las tablas de cuenta, saldo y operación en todo el proceso.
- Read (verde):** Es la operación más rápida, ya que oscila entre 400 a 500 durante toda la fase. Esto es rápido ya que no bloquea las tablas al momento de consultar. Teniendo un punto máximo de 516ms en el tiempo 18:34.
- Update (amarillo):** Presentan el tiempo de respuesta más alto, estabilizándose entre 1300 y 1800ms, con su pico máximo de 2067ms en el tiempo 18:34 de la prueba, esto debido que busca los registros en cada tabla, realiza la modificación y guardado del registro en las tablas teniendo una mayor sobrecarga transaccional.

- d) **Delete(azul):** Es el proceso que se ubica en una estabilización de 600 a 1050, un valor estable ya que se encuentra en tiempos menores de create y update ya que este proceso se encarga de buscar y eliminar el registro, teniendo un tiempo de respuesta más alto en 1253 en el tiempo 18:34.

Figura 73

Gráfica de tiempo de respuesta durante la prueba



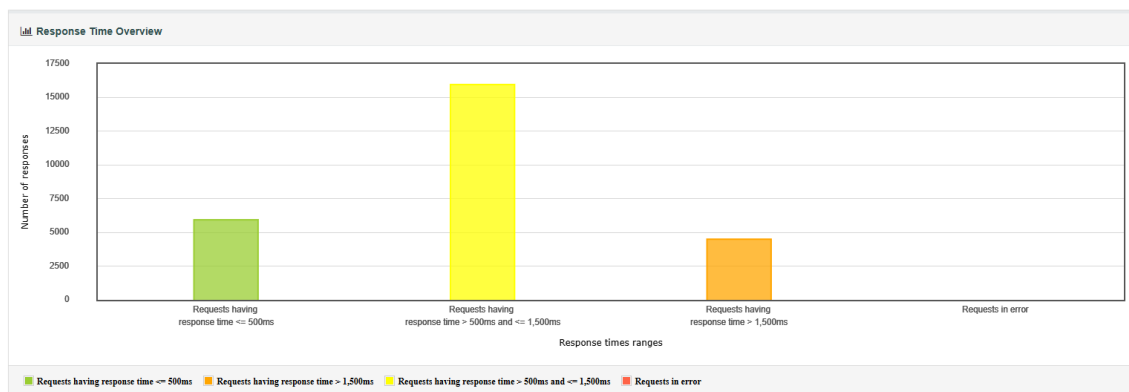
b) Distribución de cantidad de peticiones y tiempo de respuesta

En la siguiente figura 74 se muestra la distribución general de cantidades de respuestas, agrupando según el rango de tiempo que duro la peticiones se dividen en 3 partes:

- Tiempo de respuesta de ≤ 500 ms: Se 5939 peticiones
- Tiempo de respuesta entre $500 < > 1500$: 15964 peticiones
- Tiempo de respuesta mayor de >1500 : 4496 peticiones

Figura 74

Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo sincrónico



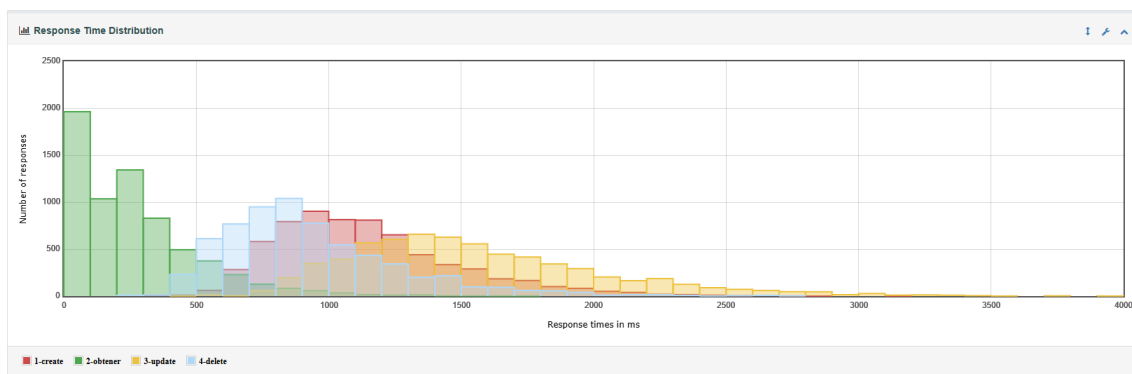
c) Distribución de tiempo de respuesta

En la siguiente figura 75 se muestra la distribución de frecuencias de tiempo de respuesta, separada por operación CRUD.

- a) **Create (rojo):** Se distribuye en mayor concentración en 800 a 1100ms, teniendo la mayor distribución en el rango de 903 en 900 a 1000ms.
- b) **Read (verde):** Varias peticiones se concentran en el rango de 0 a 500ms, teniendo en la mayor distribución de 1964 peticiones en 0 a 100 ms.
- c) **Update (amarillo):** Se distribuye en mayor concentración en 1100 a 1600ms, teniendo la mayor distribución con 659 peticiones en 1300 a 1400ms.
- d) **Delete(azul):** Se distribuye principalmente en 600 a 1000ms, teniendo en la mayoría 1040 peticiones en 800 a 900ms.

Figura 75

Gráfica de distribución de tiempo de respuesta



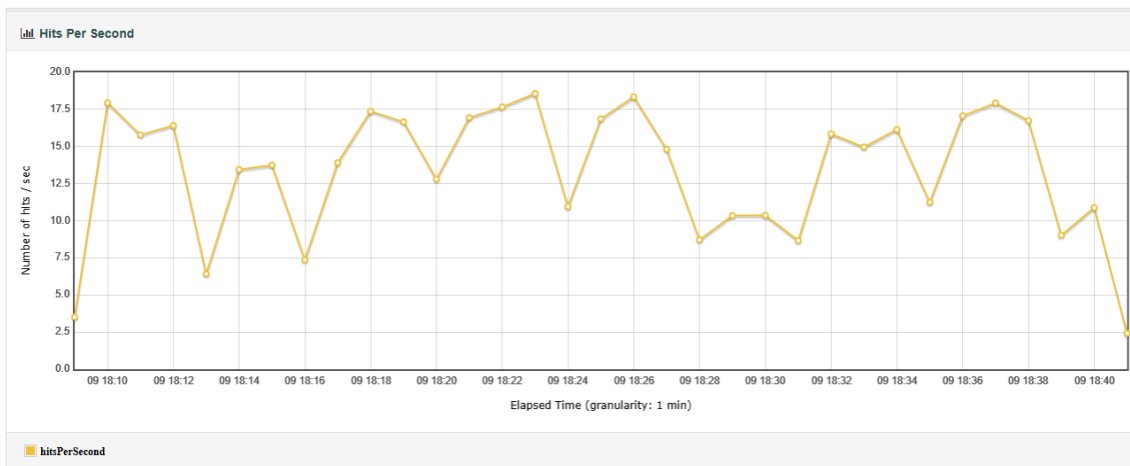
4.5.3.1.3 Throughput

a) TPS en general

En la siguiente figura 76 se puede observar el total de los TPS en el transcurso de la prueba como va variando en todo el transcurso, en este caso es la suma general de las 4 operaciones CRUD, se puede observar que hay un ascenso cuando empieza en los primeros minutos es normal ya que varios usuarios están siendo activados para realizar peticiones y dentro de algunos minutos tiene caídas y nuevamente teniendo un ascenso y nuevamente teniendo un descenso ya que usuarios se van disminuyendo, llegando a su punto máximo 18.57 de TPS en el tiempo 18.23.

Figura 76

Gráfica de TPS en general del aplicativo sincrónico



b) TPS por operación

En la siguiente figura 77 se muestra el throughput (transacciones por segundo) de las cuatro operaciones CRUD a lo largo del tiempo de prueba. Acá se observa que las 4 operaciones se comportan de manera prácticamente idénticas.

- a) **Create (azul):** Se puede observar que alcanza su punto máximo de 4.7 en el tiempo de 18:23.
- b) **Read (amarillo):** Se puede observar que alcanza su punto máximo de 4.7 en el tiempo de 18:23.
- c) **Update (rojo):** Se puede observar que alcanza su punto máximo de 4.7 en el tiempo de 18:23.
- d) **Delete(verde):** Se puede observar que alcanza su punto máximo de 4.7 en el tiempo de 18:23.

Figura 77

Gráfica del TPS por operación del aplicativo sincrónico



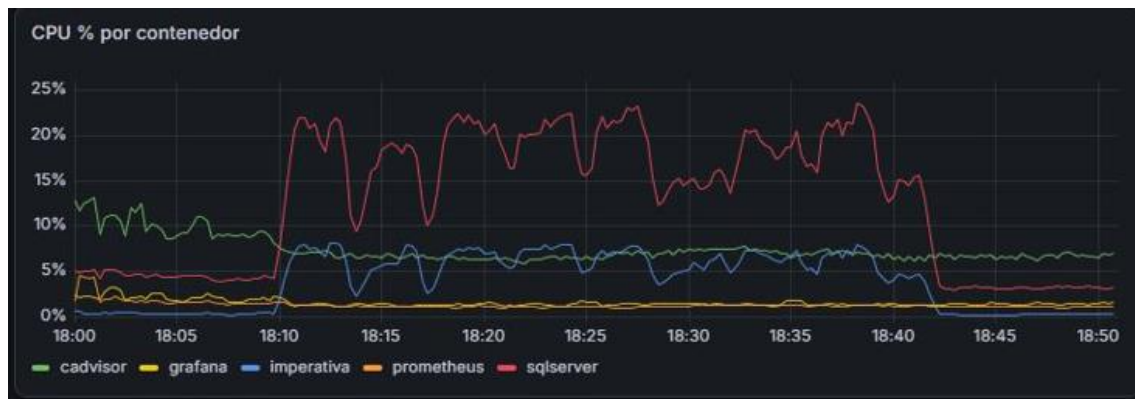
4.5.3.1.4 Consumo de recurso

a) Consumo de CPU

En la siguiente figura 78 se observa el consumo de CPU por contenedor Docker durante los 30 minutos de prueba. El contenedor de SQL Server (rojo) presenta el consumo más alto, alcanzando picos de aproximadamente 23% durante varias fases estables, mientras que el contenedor de la aplicación imperativa (azul) se mantiene en un rango de 5-8%, con un pico de 9%. Los contenedores de monitoreo (cAdvisor, Grafana, Prometheus) presentan un consumo pequeño, por debajo del 5%, confirmando que no interfieren en la medición.

Figura 78

Gráfica del consumo de CPU del aplicativo sincrónico

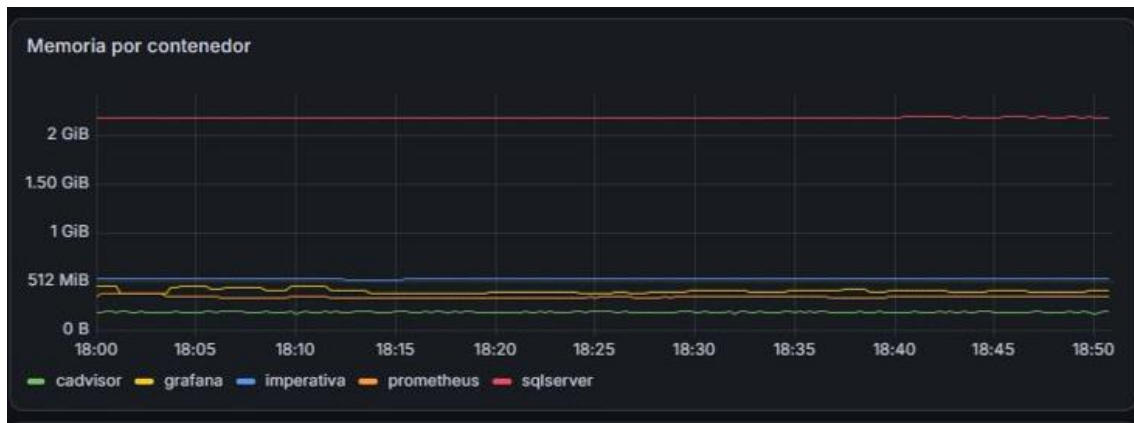


b) Consumo de memoria

En la siguiente figura 79 se muestra la memoria se mantiene prácticamente constante durante toda la prueba, sin picos ni variaciones asociadas al aumento de carga. El contenedor de SQL Server (rojo) consume de forma estable 2.3 GiB, mientras que la aplicación imperativa (azul) se mantiene alrededor de 400-512 MiB.

Figura 79

Gráfica de consumo de memoria del aplicativo sincrónico



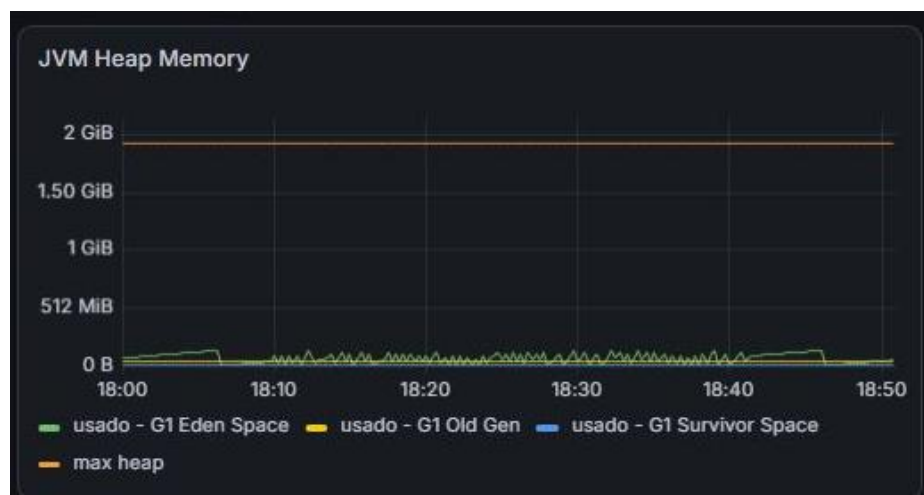
c) Consumo de JVM

1. JVM Heap Memory

En la siguiente figura 80 se muestra el uso del heap de la JVM de manera desagregado por generación: G1 Eden Space, G1 Old Gen y G1 Survivor Space, junto con el límite máximo configurado. Los tres componentes se mantienen en valores estables durante toda la prueba, lo que indica que la aplicación está operando normal sin alcanzar su capacidad máxima de heap asignada. Esto significa que, para esta carga de prueba con 20 usuarios concurrentes, la memoria no constituye un factor que limite el rendimiento observado, y que existe un amplio margen antes de que la gestión de memoria se convierta en un cuello de botella.

Figura 80

Gráfica de consumo de JVM Heap Memory del aplicativo sincrónico

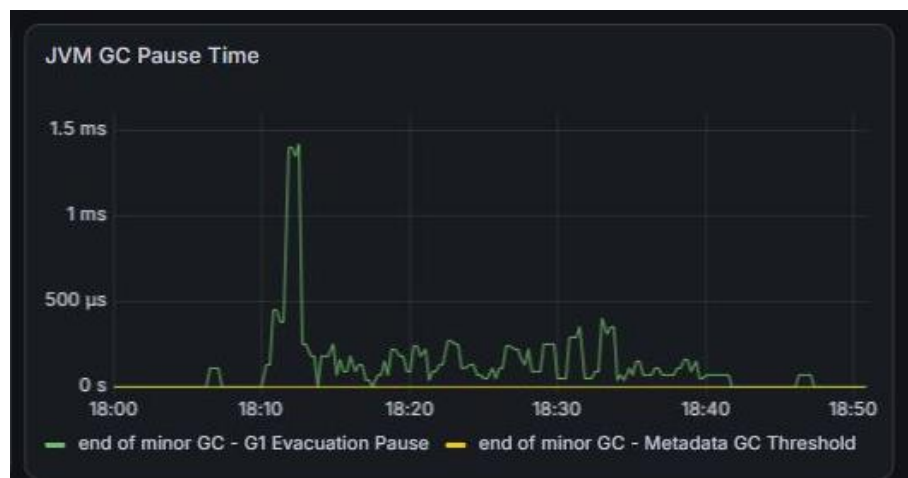


2. JVM GC Pause Time

En la siguiente figura 81 se observa el tiempo de pausa generado por el recolector de basura (Garbage Collector) durante la ejecución. Las pausas se mantienen mayormente por debajo de 500 microsegundos a lo largo de la prueba, con un pico de aproximadamente 1.5 ms alrededor del 18:15, coincidiendo con el inicio de la prueba. El comportamiento del GC es regular y de bajo impacto, lo que indica que la gestión de memoria de la JVM no representa una fuente significativa de latencia adicional para las operaciones CRUD evaluadas.

Figura 81

Gráfica de consumo de JVM GC Pause Time del aplicativo sincrónico



3. JVM Threads

En la siguiente figura 82 se muestra el número de hilos vivos (verde) e hilos daemon (amarillo) de la JVM. Se observa un incremento desde 20 hilos en reposo hasta estabilizarse entre 35-38 hilos activos durante la fase de carga en el tiempo (18:10-28:40), y luego un descenso a los valores base al finalizar la prueba. Este comportamiento es el esperado, donde la petición HTTP entrante ocupa un hilo dedicado durante toda su ejecución de procesamiento + espera de respuesta de la base de datos. El número de hilos supera ese valor ya que tiene valor de 35-38 esto refleja un adicional de hilos internos del framework y del contenedor de aplicaciones Tomcat y no solo los hilos dedicados directamente a las peticiones de usuario.

Figura 82

Gráfica de consumo de JVM Threads del aplicativo sincrónico

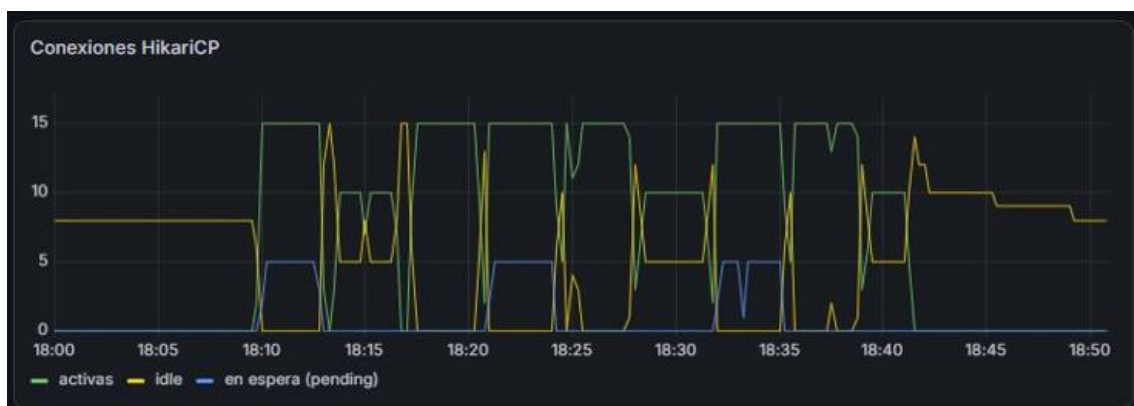


d) Conexiones HikariCP

En la siguiente figura 83 se muestra el estado del pool de conexiones a la base de datos, conexiones activas (verde), idle/libres (amarillo) y en espera/pending (azul). Se observa que, al iniciar la carga en el tiempo 18:10, las conexiones activas alcanzan rápidamente su máximo configurado de 15 conexiones y se mantienen en ese tope durante toda la fase, mientras que simultáneamente aparecen conexiones en espera (pending), estabilizándose en aproximadamente 5. Esto indica que el pool de conexiones se saturó bajo la carga de 20 usuarios concurrentes, y que existían peticiones adicionales esperando a que se liberé una conexión antes de poder ejecutar la siguiente operación contra la base de datos generando un cuello de botella directamente y esto es atribuible al tamaño configurado del pool y al modelo de conexión bloqueante, donde cada hilo retiene su conexión durante todo el ciclo de vida de la petición.

Figura 83

Gráfica de conexiones de HikariCP del aplicativo sincrónico



e) Tiempo de espera de conexión

En la siguiente figura 84 se muestra el tiempo que una petición debe esperar para obtener una conexión libre del pool. Se observa un salto de 0 ms a aproximadamente 5 segundos constantes entre los minutos 18:10, 18:20 y 18:33. Esto significa que, durante gran parte de la prueba, una porción significativa del tiempo total de respuesta de la aplicación no correspondía al procesamiento real de la operación CRUD, sino a la espera por obtener una conexión disponible hacia SQL Server.

Figura 84

Gráfica de tiempo de espera de conexión del aplicativo sincrónico



f) Throughput SQL Server

En la siguiente figura 85 se muestran dos métricas batch requests/sec (verde), que mide lotes de comandos SQL ejecutados, y transacciones/sec (amarillo). Ambas curvas muestran un crecimiento sostenido desde el inicio de la prueba hasta estabilizarse alrededor del minuto 18:50, alcanzando aproximadamente 1.25Mil batch requests acumulados y 150K transacciones acumuladas.

Figura 85

Gráfica de Throughput del consumo de SQL server del aplicativo sincrónico



g) Locks y deadlocks

En la siguiente figura 86 se observa un crecimiento sostenido de los lock waits durante la fase de carga, alcanzando aproximadamente 22.5K esperas acumuladas antes de estabilizarse, mientras que la línea de deadlocks se mantiene prácticamente en 0 durante toda la prueba. Esto indica que, si existieron esperas por bloqueos de fila/tabla y no se registraron deadlocks que hubieran obligado a SQL Server a abortar transacciones, lo cual es confiable con la tasa de error del 0% observada en JMeter.

Figura 86

Gráfica de lock y deadlock del aplicativo sincrónico



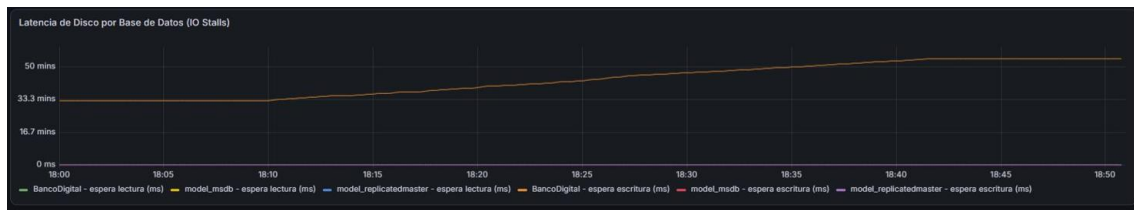
h) Latencia de Disco por base de datos

En la siguiente figura 87 se mide el tiempo acumulado de espera en operaciones de lectura/escritura a nivel de disco físico. La base de datos BancoDigital, para realizar una escritura (naranja) presenta el crecimiento más alto, acumulando hasta aproximadamente 33.3 minutos de espera total de I/O de escritura hacia el final de la prueba, mientras que las esperas de lectura y las de las bases de datos del sistema (model_msdb, model_replicatedmaster) permanecen cercanas a 0. Esto significa que las operaciones

CRUD con proporción significativa de escrituras: create, update, delete, lo cual es coherente la mayor parte de la latencia a nivel de disco proviene específicamente de las operaciones de escritura sobre la base de datos de la aplicación, no de operaciones de lectura ni de procesos internos del motor de base de datos.

Figura 87

Gráfica de latencia del disco del aplicativo sincrónico



4.5.3.1.5 Estadística descriptiva de la prueba de picos del aplicativo sincrónico

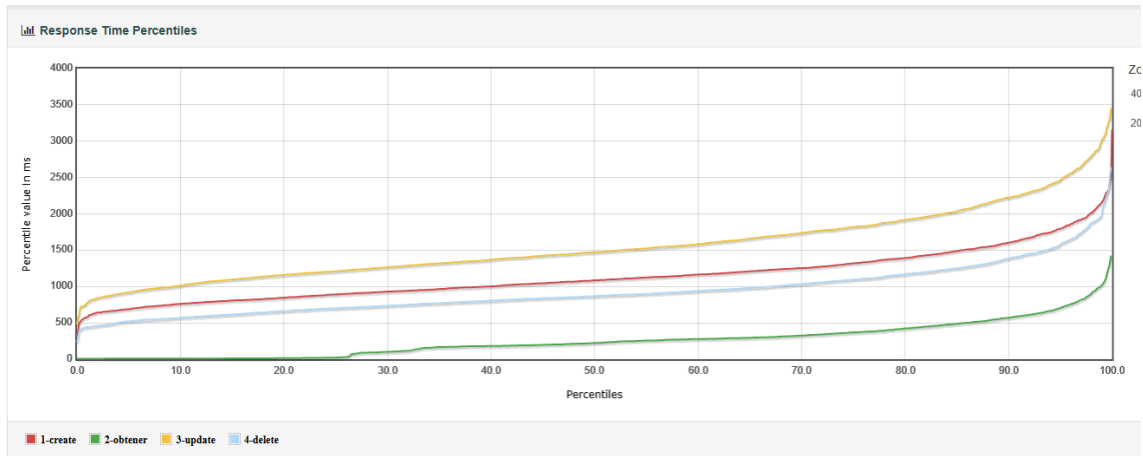
En la siguiente figura 88 se muestra el tiempo de respuesta acumulado por percentil para cada operación CRUD:

- Create (rojo):** muestra una mediana de aproximadamente 1150 ms y un percentil 90 de aproximadamente 1650 ms.
- Read(verde):** mantiene los valores más bajos en todo el rango, con un percentil 50 (mediana) de aproximadamente 250-300 ms y un percentil 90 cercano a 550-600 ms.
- Update (amarillo):** presenta los valores más altos en todos los percentiles, con una mediana de aproximadamente 1750 ms y un percentil 90 cercano a 2300 ms.
- Delete (celeste):** presenta una mediana de aproximadamente 900 ms y un percentil 90 cercano a 1400 ms.

En el extremo superior percentil 99-100, las cuatro curvas muestran un incremento, alcanzando valores máximos de aproximadamente 3400 ms para update, 2750 ms para create, 2050 ms para delete y 2000 ms para obtener, lo que evidencia que, existe un subconjunto de solicitudes con tiempos de respuesta considerablemente más altos que el resto.

Figura 88

Gráfica de percentiles del aplicativo sincrónico



Validación de los resultados

a) Tasa de error

Se registró una tasa de error del 0.00% en las cuatro operaciones evaluadas sobre un total de 26 425 mediciones, lo que indica que el aplicativo no presentó fallos funcionales durante toda la prueba, sin fallos, excepciones ni interrupciones de conexión registradas en ninguna de las operaciones.

- a) **Create:** 6648 ejecuciones, 0 fallos (0.00%)
- b) **Read:** 6606 ejecuciones, 0 fallos (0.00%)
- c) **Update:** 6597 ejecuciones, 0 fallos (0.00%)
- d) **Delete:** 6548 ejecuciones, 0 fallos (0.00%)

b) Error estándar de la media

El error estándar de la media ($SE = \text{desviación estándar} / \sqrt{n}$) resulta pequeño en relación con el promedio de cada operación, lo que indica que los valores constituyen estimaciones precisas y confiables del verdadero comportamiento del sistema, esto mide cuando variaría el promedio si repitiera nuevamente la prueba.

- a) **Create:**
 - a. $SE = 340.55 / \sqrt{6648} = 4.17$ ms sobre un promedio de 1137 ms = 0.36%
- b) **Read:**
 - a. $SE = 234.89 / \sqrt{6606} = 2.87$ ms sobre un promedio de 259 ms = 1.1%
- c) **Update:**
 - a. $SE = 477.46 / \sqrt{6597} = 5.87$ ms sobre un promedio de 1555 ms = 0.37%
- d) **Delete:**

$$SE = 336.00 / \sqrt{6548} = 4.15 \text{ ms sobre un promedio de } 933 \text{ ms} = 0.44\%$$

c) Cercanía entre promedio y mediana

Se evidencia una cercanía similar entre el promedio y la mediana en las cuatro operaciones evaluadas, con diferencias mínimas en todos los casos, lo que confirma que los resultados no se encuentran distorsionados por valores extremos y que la distribución de tiempos de respuesta es relativamente simétrica.

- a) **Create:** promedio 1137 ms frente a mediana 1086 ms (diferencia de 4.48%)
- b) **Read:** promedio 259.18 ms frente a mediana 223 ms (diferencia de 13.89%)
- c) **Update:** promedio 1555.67 ms frente a mediana 1472 ms (diferencia de 5.37%)
- d) **Delete:** promedio 933.38 ms frente a mediana 868 ms (diferencia de 7.0%)

d) Progresión gradual de percentiles

En las cuatro operaciones se observa un incremento progresivo y sin saltos abruptos entre los percentiles 50, 90, 95 y 99, lo que descarta la presencia de comportamientos erráticos o bimodales en el sistema durante la prueba.

- a) **Create:** mediana 1086 ms, → 90% 1603 ms, → 95% 1797 ms, → 99% 2154 ms
- b) **Read:** mediana 223 ms, → 90% 570 ms, → 95% 705 ms, → 99% 1021 ms
- c) **Update:** mediana 1472 ms, → 90% 2222 ms, → 95% 2462 ms, → 99% 3005 ms
- d) **Delete:** mediana 868 ms, → 90% 1386 ms, → 95% 1572 ms, → 99% 2000 ms

e) Throughput similar entre operaciones

El throughput se mantuvo prácticamente idéntico entre las cuatro operaciones, lo que evidencia un procesamiento predecible y balanceado del sistema bajo la carga aplicada, independientemente del tiempo de respuesta individual de cada operación.

- a) **Create:** 3.5 transacciones/seg
- b) **Read:** 3.5 transacciones/seg
- c) **Update:** 3.5 transacciones/seg
- d) **Delete:** 3.5 transacciones/seg

f) Volumen y duración suficientes para diluir el ruido de corto plazo

El volumen total de mediciones recolectadas durante los 30 minutos de prueba continua resulta suficiente para que efectos puntuales de corta duración.

- e) **Create:** 6648 mediciones
- f) **Read:** 6606 mediciones
- g) **Update:** 6597 mediciones
- h) **Delete:** 6548 mediciones

Total: 26 399 mediciones en 30 minutos de prueba continua

4.5.3.2 Resultados del aplicativo asincrónico

Los resultados que se tuvo para el aplicativo asincrónico fueron los siguientes:

Tabla 12

Resultados del aplicativo asincrónico en la prueba de picos

Endpoint	Cantida d de ejecucio nes	Tiemp o de respues ta promedio (ms)	Tiem po de míni mo de respu esta (ms)	Tiemp o de máxim o de respu esta (ms)	Desviaci ón estándar	Through put	Medi ana	90%.	95%
Creación	7561	1388	322	3335	417.13	4.0/sec.	1301	1983	2167
Obtener	7505	77	8	821	110.79	4.0/sec.	25	251	319
Edición	7502	854	184	2564	365.34	4.0/sec.	776	1402	1619
Eliminación	7470	1089	272	2704	353.97	3.9/sec.	1019	1591	1765
Total	30038	853	8	3335	589.78	15.8/sec.	882	1632	1857

Nota. Elaboración propia.

Se detalla a continuación sobre cada endpoint sus métricas:

- a) **Resultado del endpoint Creación:** Se ejecutó 7561 veces, con un promedio de respuesta de 1388 ms, el mínimo tiempo de respuesta fue de 322 ms y el máximo de 3335 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 417.13, con un throughput de 4.0/sec., con una mediana de 1301 y percentil 90 de 1983 y percentil 95 de 2167 en JMeter.

- b) Resultado del endpoint Obtener:** Se ejecutó 7505 veces, con un promedio de respuesta de 77 ms, el mínimo tiempo de respuesta fue de 8 ms y el máximo de 821 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 110.79, con un throughput de 4.0/sec., con una mediana de 25 y percentil 90 de 251 y percentil 95 de 319 en JMeter.
- c) Resultado del endpoint Edición:** Se ejecutó 7502 veces, con un promedio de respuesta 854 ms, el mínimo tiempo de respuesta fue de 184 ms y el máximo 2564 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) de 365.34, con un throughput de 4.0/sec., con una mediana de 776 y percentil 90 de 1402 y percentil 95 de 1619 en JMeter.
- d) Resultado del endpoint Eliminación:** Se ejecutó 7470 veces, con un promedio de respuesta de 1089 ms, el mínimo tiempo de respuesta fue de 272 ms y el máximo de 2704 ms, con una desviación estándar (mide la dispersión que hay en el tiempo de respuesta y promedio) 353.97, con un throughput de 3.9/sec., con una mediana de 1019 y percentil 90 de 1591 y percentil 95 de 1765 en JMeter.

4.5.3.2.1 Latencia

a) Latencia durante la prueba

En la siguiente figura 89 se muestra la latencia promedio en el tiempo transcurrido de la prueba para las 4 operaciones CRUD. El gráfico muestra latencia promedio vs el tiempo de la prueba, en la cual mide específicamente el tiempo hasta recibir el primer byte de respuesta del servidor.

Se observa un comportamiento similar para cada operación donde hay una fase de ascenso esto sería normal ya que en los primeros minutos de la prueba va creciendo los usuarios hasta llegar a los 20 usuarios, seguida de una fase de estable de unos minutos y un descenso y así sucesivamente hasta terminar la prueba.

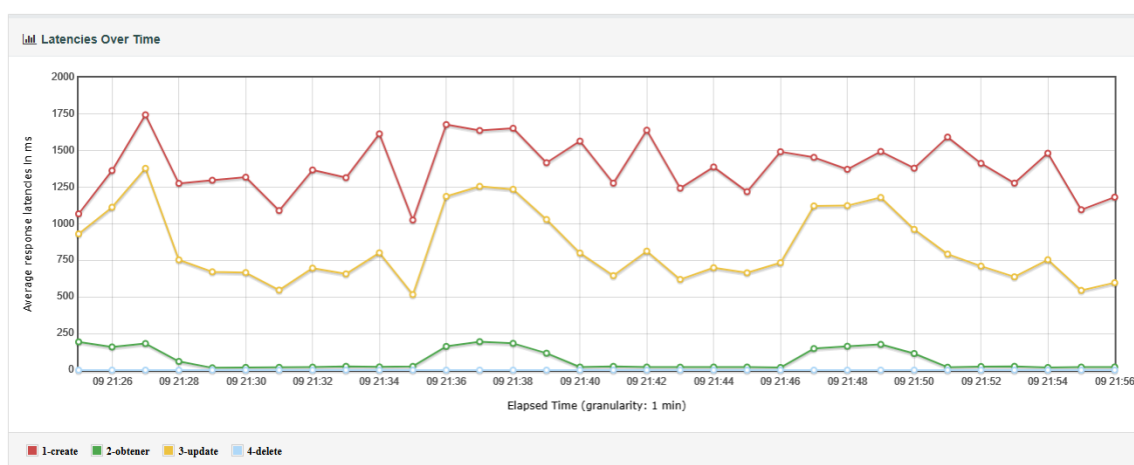
Por operación se detalla lo siguiente:

- d) Create (rojo):** Se ubica en un rango intermedio, estabilizándose entre 1250 a 1600ms con un pico máximo de 1746 en el minuto 21:27.
- e) Read (verde):** Se muestra una latencia considerablemente menor con respecto a los endpoint de create y update, estabilizándose entre 0 a 250ms, esto normal ya que las operaciones de lectura son más rápidas.

- f) **Update (amarillo):** Presenta una latencia estable, estabilizándose entre 500 a 1200ms, con un pico de máximo de 1380 en el minuto 21:27.
- g) **Delete (azul):** Se mantiene en 0ms durante la prueba ya que para esta operación no se espera ni una respuesta(response) que tenga contenido, es por eso que la respuesta no tiene un byte para poder calcularlo.

Figura 89

Gráfica de latencia del aplicativo asincrónico



4.5.3.2.2 Response Time

a) Tiempo de respuesta durante la prueba

En el siguiente gráfico 90 se muestra el tiempo de respuesta promedio en el tiempo que duro la prueba, se puede observar el grafico de las 4 operaciones (CRUD). El tiempo de respuesta mide el proceso completo de la solicitud, desde el envío hasta la recepción total de la respuesta.

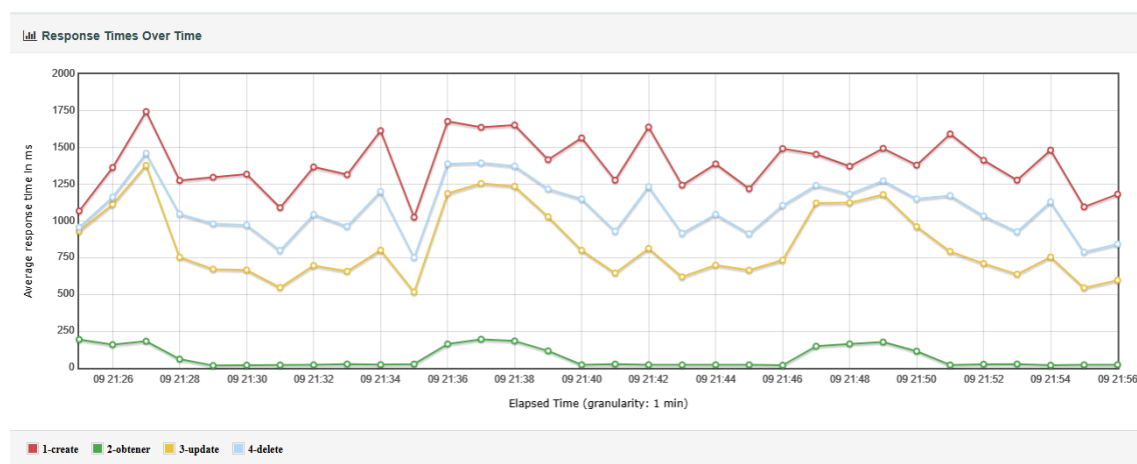
En el gráfico se identifican también 3 fases que la primera es la de creciente o ascenso en la cual se puede observar que en los primeros minutos una elevación ya que en esos momentos se están incrementando los usuarios hasta llegar hasta los 20 usuarios, después se mantiene una estabilidad de unos minutos ya que se encuentra activos los 20 usuarios, después una fase de descenso en la cual los usuarios van decreciendo y así sucesivamente hasta terminar la prueba. Se detalla por operación:

- a) **Create (rojo):** Se ubica en un rango de 1250 a 1600, con el punto más alto en 1746 en el tiempo 21:27. Estos comportamientos es propio de una operación de escritura que requiere insertar registros a las tablas de cuenta, saldo y operación en todo el proceso.

- b) **Read (verde):** Es la operación más rápida, ya que oscila entre 0 a 250 durante toda la fase. Esto es rápido ya que no bloquea las tablas al momento de consultar. Teniendo un punto máximo de 195ms en el tiempo 21:27.
- c) **Update (amarillo):** Presentan el tiempo de respuesta más alto, estabilizándose entre 500 y 1200ms, con su pico máximo de 1380ms en el tiempo 21:27 de la prueba, esto debido que busca los registros en cada tabla, realiza la modificación y guardado del registro en las tablas teniendo una mayor sobrecarga transaccional.
- d) **Delete(azul):** Es el proceso que se ubica en una estabilización de 900 a 1300, un valor estable ya que se encuentra en tiempos menores de create y update ya que este proceso se encarga de buscar y eliminar el registro, teniendo un tiempo de respuesta más alto en 1462 en el tiempo 21:27.

Figura 90

Gráfica de tiempo de respuesta durante la prueba



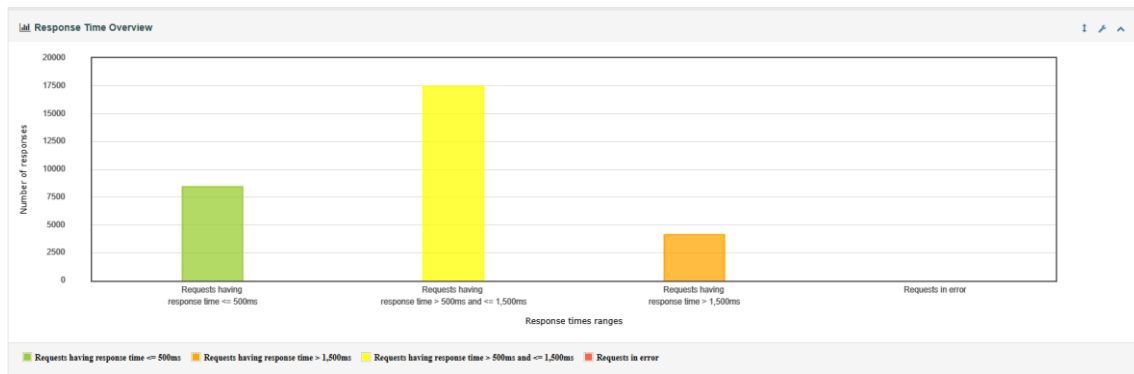
b) Distribución de cantidad de peticiones y tiempo de respuesta

En la siguiente figura 91 se muestra la distribución general de cantidades de respuestas, agrupando según el rango de tiempo que duro la peticiones se dividen en 3 partes:

- a) Tiempo de respuesta de ≤ 500 ms: Se 8432 peticiones
- b) Tiempo de respuesta entre $500 < > 1500$: 17476 peticiones
- c) Tiempo de respuesta mayor de >1500 : 4130 peticiones

Figura 91

Gráfica de distribución de cantidad de tiempo de respuesta del aplicativo asincrónico



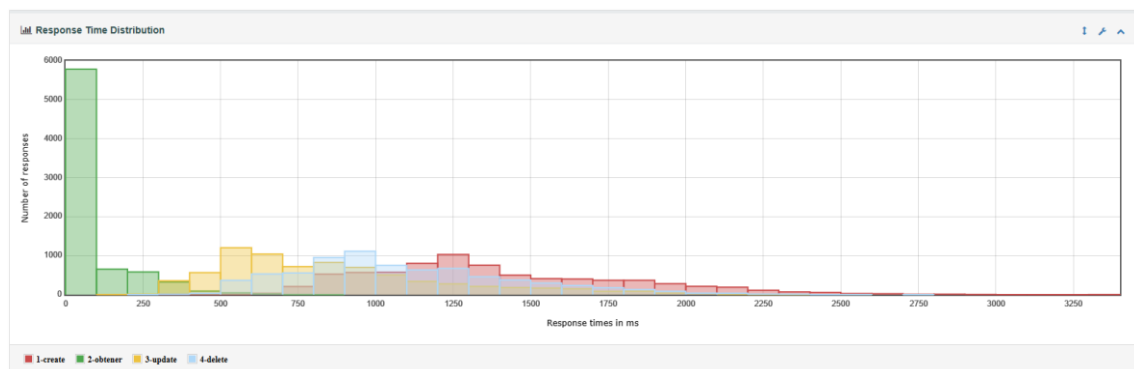
c) Distribución de tiempo de respuesta

En la siguiente figura 92 se muestra la distribución de frecuencias de tiempo de respuesta, separada por operación CRUD.

- a) **Create (rojo):** Se distribuye en mayo concentración en 800 a 1500ms, teniendo la mayor distribución en el rango de 1029 en 1200 a 1300ms.
- b) **Read (verde):** Varias peticiones se concentran en el rango de 0 a 100ms, teniendo en la mayor distribución de 5782 peticiones en 0 a 100 ms.
- c) **Update (amarillo):** Se distribuye en mayor concentración en 500 a 1000ms, teniendo la mayor distribución con 1203 peticiones en 500 a 600ms.
- d) **Delete(azul):** Se distribuye principalmente en 700 a 1300ms, teniendo en la mayoría 1112 peticiones en 900 a 1000ms.

Figura 92

Gráfica de distribución de tiempo de respuesta



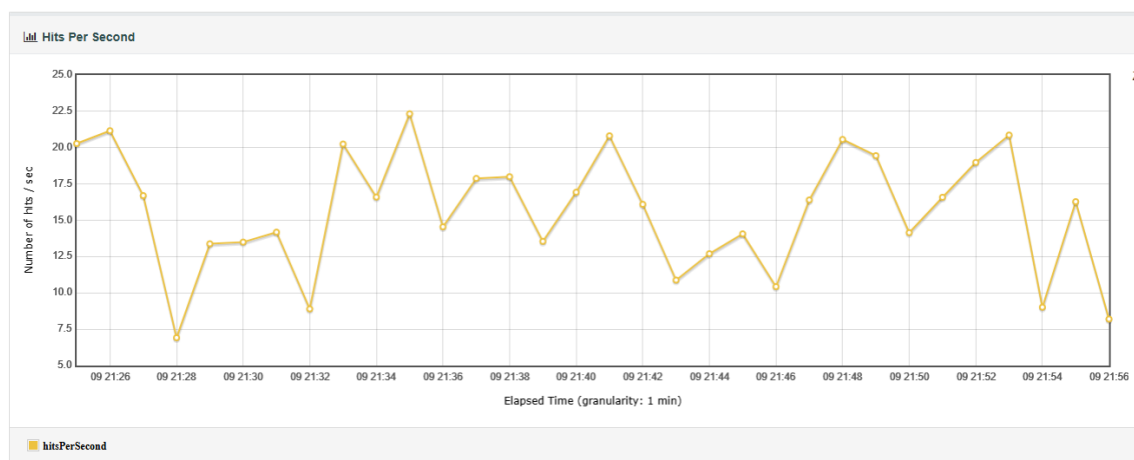
4.5.3.2.3 Throughput

a) TPS en general

En la siguiente figura 93 se puede observar el total de los TPS en el transcurso de la prueba como va variando en todo el transcurso, en este caso es la suma general de las 4 operaciones CRUD, se puede observar que hay un ascenso cuando empieza en los primeros minutos es normal ya que varios usuarios están siendo activados para realizar peticiones y dentro hay varios picos y bajadas ya que la prueba es de picos y se muestra por eso una inestabilidad controlada, alcanzando a su punto máximo 22.33 en el tiempo 21:35.

Figura 93

Gráfica de TPS en general del aplicativo asincrónico



b) TPS por operación

En la siguiente figura 94 se muestra el throughput (transacciones por segundo) de las cuatro operaciones CRUD a lo largo del tiempo de prueba. Acá se observa que las 4 operaciones se comportan de manera prácticamente identificas.

- a) **Create (azul):** Se puede observar que alcanza su punto máximo de 5.7 en el tiempo de 21:35.
- b) **Read (amarillo):** Se puede observar que alcanza su punto máximo de 5.7 en el tiempo de 21:35.
- c) **Update (rojo):** Se puede observar que alcanza su punto máximo de 5.7 en el tiempo de 21:35.
- d) **Delete(verde):** Se puede observar que alcanza su punto máximo de 5.7 en el tiempo de 24:35.

Figura 94

Gráfica del TPS por operación del aplicativo asincrónico



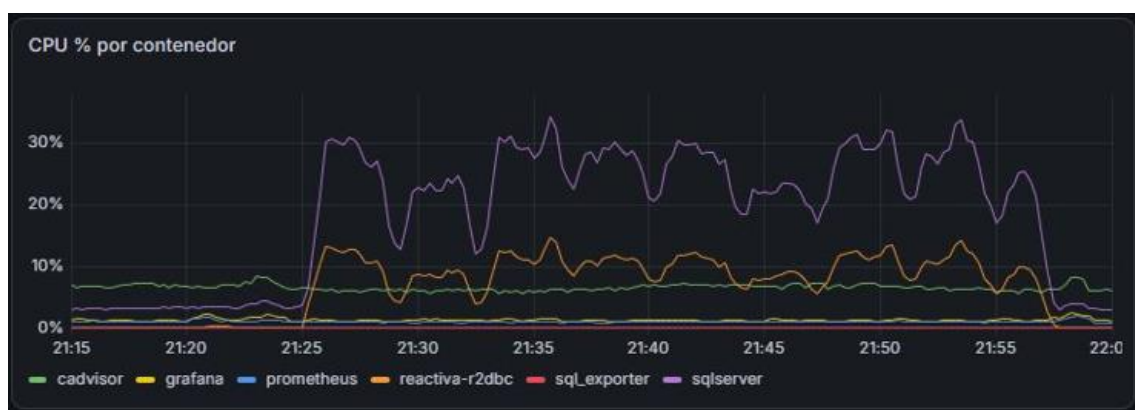
4.5.3.2.4 Consumo de recursos

a) Consumo de CPU

En la siguiente figura 95 se observa el consumo de CPU por contenedor Docker durante los 30 minutos de prueba. El contenedor de SQL Server (morado) presenta el consumo más alto, alcanzando picos de aproximadamente 32-33% durante varias, mientras que el contenedor de la aplicación reactivo (naranja) se mantiene en un rango de 8-15%, con un pico de 15% al inicio de la prueba (21:38). Los contenedores de monitoreo (cAdvisor, Grafana, Prometheus) presentan un consumo pequeño, por debajo del 10%, confirmando que no interfieren en la medición.

Figura 95

Gráfica del consumo de CPU del aplicativo asincrónico



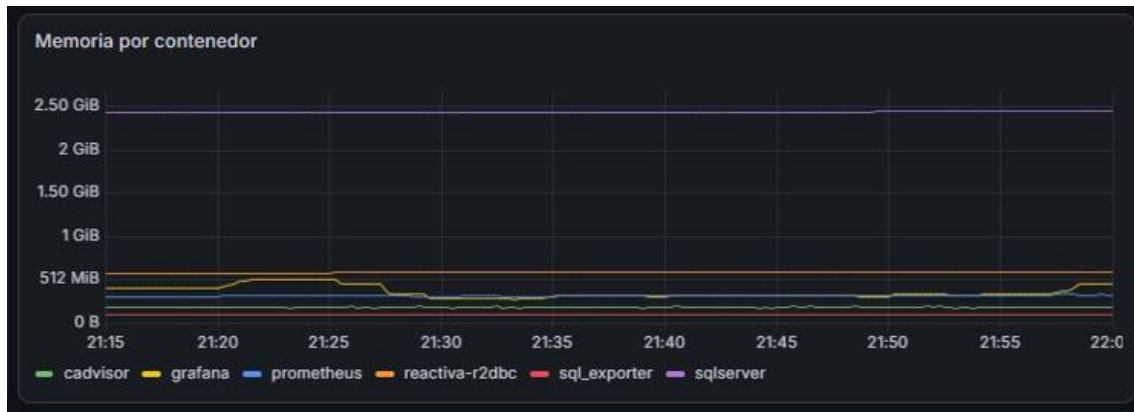
b) Consumo de memoria

En la siguiente figura 96 se muestra la memoria se mantiene prácticamente constante durante toda la prueba, sin picos ni variaciones asociadas al aumento de carga. El

contenedor de SQL Server (morado) consume de forma estable 2.5 GiB, mientras que la aplicación reactiva (naranja) se mantiene alrededor de 512 MiB.

Figura 96

Gráfica de consumo de memoria del aplicativo asincrónico



c) Consumo de JVM

1 JVM Heap Memory

En la siguiente figura 97 se muestra el uso del heap de la JVM de manera desagregado por generación: G1 Eden Space, G1 Old Gen y G1 Survivor Space, junto con el límite máximo configurado. Los tres componentes se mantienen en valores estables durante toda la prueba, lo que indica que la aplicación está operando normal sin alcanzar su capacidad máxima de heap asignada. Esto significa que, para esta carga de prueba con 20 usuarios concurrentes, la memoria no constituye un factor que limite el rendimiento observado, y que existe un amplio margen antes de que la gestión de memoria se convierta en un cuello de botella.

Figura 97

Gráfica de consumo de JVM Heap Memory del aplicativo asincrónico

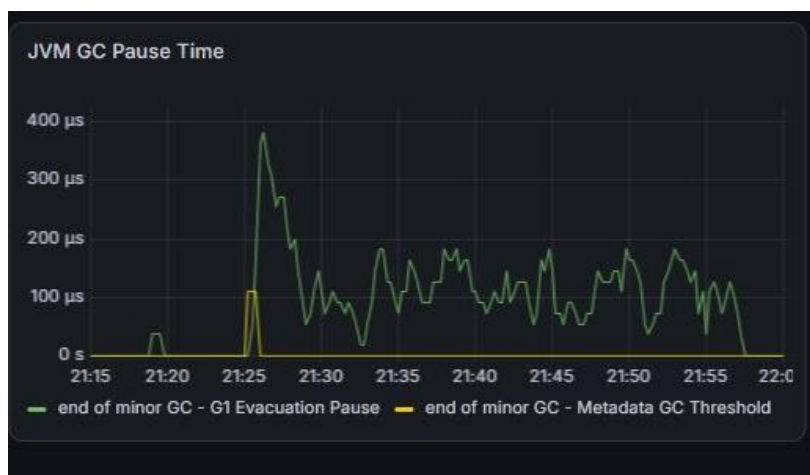


2 JVM GC Pause Time

En la siguiente figura 98 se observa el tiempo de pausa generado por el recolector de basura (Garbage Collector) durante la ejecución. Las pausas se mantienen mayormente por debajo de 200 microsegundos a lo largo de la prueba, con un pico de aproximadamente 390 microsegundos alrededor del minuto 21:28, coincidiendo con el inicio de la prueba. El comportamiento del GC es regular y de bajo impacto, lo que indica que la gestión de memoria de la JVM no representa una fuente significativa de latencia adicional para las operaciones CRUD evaluadas.

Figura 98

Gráfica de consumo de JVM GC Pause Time del aplicativo asincrónico



3 JVM Threads

En la siguiente figura 99 se muestra el número de hilos vivos (verde) e hilos daemon (amarillo) de la JVM. Se observa un incremento de 32 hilos activos

durante la fase de la prueba. Este comportamiento es el esperado, donde la petición HTTP entrante ocupa un hilo dedicado durante toda su ejecución de procesamiento + espera de respuesta de la base de datos. El número de hilos supera ese valor ya que tiene valor de 32 esto refleja un adicional de hilos internos del framework y del contenedor de aplicaciones Tomcat y no solo los hilos dedicados directamente a las peticiones de usuario.

Figura 99

Gráfica de consumo de JVM Threads del aplicativo asincrónico



d) Conexiones R2DBC

En la siguiente figura 100 se muestra el estado del pool de conexiones a la base de datos, conexiones activas (verde), idle/libres (amarillo) y en espera/pending (azul). Se observa que, al iniciar la carga en el tiempo 21:25, las conexiones activas alcanzan rápidamente su máximo configurado de 15 conexiones y se mantienen en varios picos, mientras que simultáneamente aparecen conexiones en espera (pending), no teniendo una estabilización ya que hay muchas conexiones adquiridas y liberadas es por eso que muchas conexiones están en espera. Esto indica que el pool de conexiones se saturó bajo la carga de 20 usuarios concurrentes, y que existían peticiones adicionales esperando a que se liberara una conexión antes de poder ejecutar la siguiente operación contra la base de datos generando un cuello de botella directamente y esto es atribuible al tamaño configurado del pool.

Figura 100

Gráfica de conexiones de R2dbc del aplicativo asincrónico



e) Throughput SQL Server

En la siguiente figura 101 se muestran dos métricas batch requests/sec (verde), que mide lotes de comandos SQL ejecutados, y transacciones/sec (amarillo). Ambas curvas muestran un crecimiento sostenido desde el inicio de la prueba hasta estabilizarse alrededor del minuto 22:00, alcanzando aproximadamente 2 mill. batch requests acumulados y 100K transacciones acumuladas.

Figura 101

Gráfica de Throughput del consumo de SQL server del aplicativo asincrónico



f) Locks y deadlocks

En la siguiente figura 102 se observa un crecimiento sostenido de los lock waits durante la fase de carga, alcanzando aproximadamente 20K esperas acumuladas antes de estabilizarse, mientras que la línea de deadlocks se mantiene prácticamente en 0 durante toda la prueba. Esto indica que, si existieron esperas por bloqueos de fila/tabla y no se registraron deadlocks que hubieran obligado a SQL Server a abortar transacciones, lo cual es confiable con la tasa de error del 0% observada en JMeter.

Figura 102

Gráfica de lock y deadlock del aplicativo asincrónico

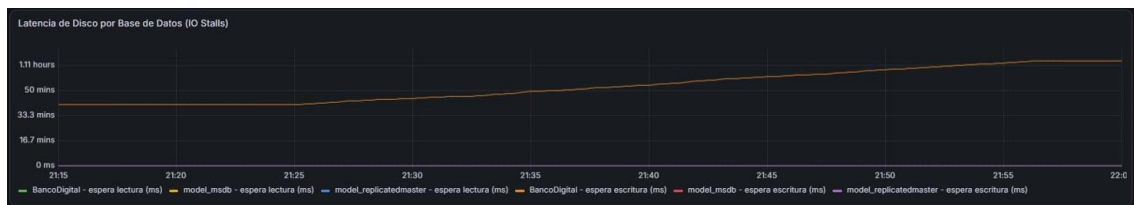


g) Latencia de Disco por base de datos

En la siguiente figura 103 se mide el tiempo acumulado de espera en operaciones de lectura/escritura a nivel de disco físico. La base de datos BancoDigital, para realizar una escritura (naranja) presenta el crecimiento más alto, acumulando hasta aproximadamente 1.11 hours de espera total de I/O de escritura hacia el final de la prueba, mientras que las esperas de lectura y las de las bases de datos del sistema (model_msdb, model_replicatedmaster) permanecen cercanas a 0. Esto significa que las operaciones CRUD con proporción significativa de escrituras: create, update, delete, lo cual es coherente la mayor parte de la latencia a nivel de disco proviene específicamente de las operaciones de escritura sobre la base de datos de la aplicación, no de operaciones de lectura ni de procesos internos del motor de base de datos.

Figura 103

Gráfica de latencia del disco del aplicativo asincrónico



4.5.3.2.5 Estadística descriptiva de la prueba de picos del aplicativo asincrónico

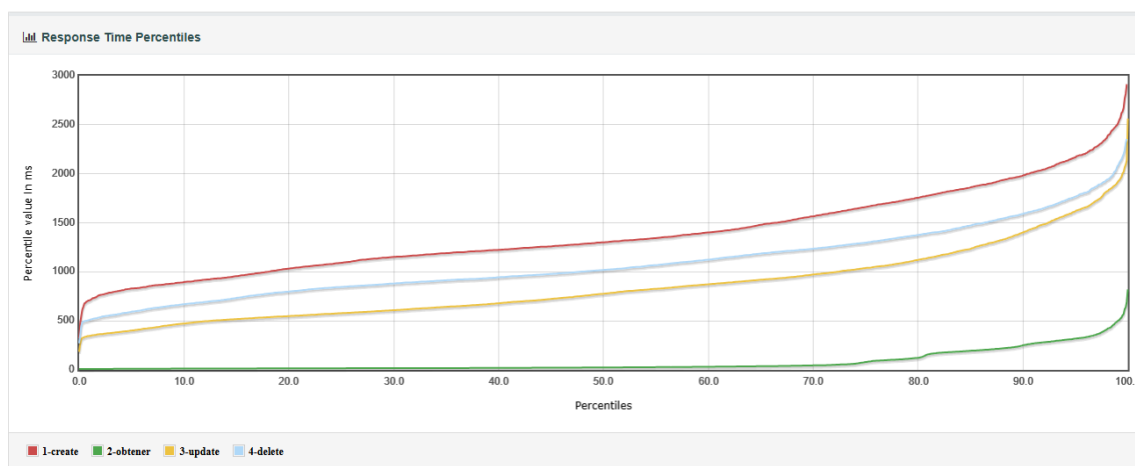
En la siguiente figura 104 se muestra el tiempo de respuesta acumulado por percentil para cada operación CRUD:

- a) **Create (rojo):** muestra una mediana de aproximadamente 1301 ms y un percentil 90 de aproximadamente 1983 ms.
- b) **Read(verde):** mantiene los valores más bajos en todo el rango, con un percentil 50 (mediana) de aproximadamente 25 ms y un percentil 90 cercano a 251 ms.
- c) **Update (amarillo):** presenta los valores más altos en todos los percentiles, con una mediana de aproximadamente 776 ms y un percentil 90 cercano a 1402 ms.
- d) **Delete (celeste):** presenta una mediana de aproximadamente 1019 ms y un percentil 90 cercano a 1632 ms.

En el extremo superior percentil 99-100, las cuatro curvas muestran un incremento, alcanzando valores máximos de aproximadamente 1911 ms para update, 2501 ms para create, 1767 ms para delete y 498 ms para obtener, lo que evidencia que, existe un subconjunto de solicitudes con tiempos de respuesta considerablemente más altos que el resto.

Figura 104

Gráfica de percentiles del aplicativo asincrónico



Validación de los resultados

a) Tasa de error

Se registró una tasa de error del 0.00% en las cuatro operaciones evaluadas sobre un total de 26 425 mediciones, lo que indica que el aplicativo no presentó fallos durante toda la prueba, sin fallos, excepciones ni interrupciones de conexión registradas en ninguna de las operaciones.

- a) **Create:** 7561 ejecuciones, 0 fallos (0.00%)

- b) **Read:** 7505 ejecuciones, 0 fallos (0.00%)
- c) **Update:** 7502 ejecuciones, 0 fallos (0.00%)
- d) **Delete:** 7470 ejecuciones, 0 fallos (0.00%)

b) Error estándar de la media

El error estándar de la media ($SE = \text{desviación estándar} / \sqrt{n}$) resulta pequeño en relación con el promedio de cada operación, lo que indica que los valores constituyen estimaciones precisas y confiables del verdadero comportamiento del sistema, esto mide cuando variaría el promedio si repitiera nuevamente la prueba.

- a) **Create:**
 - a. $SE = 417.13 / \sqrt{7561} = 4.79$ ms sobre un promedio de 1388 ms = 0.34%
- b) **Read:**
 - a. $SE = 110.79 / \sqrt{7505} = 1.26$ ms sobre un promedio de 77 ms = 1.64%
- c) **Update:**
 - a. $SE = 365.34 / \sqrt{7502} = 4.21$ ms sobre un promedio de 854 ms = 0.49%
- d) **Delete:**
 - a. $SE = 353.97 / \sqrt{7470} = 4.09$ ms sobre un promedio de 1089 ms = 0.37%

c) Cercanía entre promedio y mediana

Se evidencia una cercanía similar entre el promedio y la mediana en las cuatro operaciones evaluadas, con diferencias mínimas en todos para los 3 sin contar el Read, lo que confirma que los resultados no se encuentran distorsionados por valores extremos y que la distribución de tiempos de respuesta es relativamente simétrica.

- a) **Create:** promedio 1388 ms frente a mediana 1301 ms (diferencia de 6.2%)
- b) **Read:** promedio 77 ms frente a mediana 25 ms (diferencia de 67.1%)
- c) **Update:** promedio 854 ms frente a mediana 776 ms (diferencia de 9.13%)
- d) **Delete:** promedio 1089 ms frente a mediana 1019 ms (diferencia de 6.42%)

d) Progresión gradual de percentiles

En las cuatro operaciones se observa un incremento progresivo y sin saltos abruptos entre los percentiles 50, 90, 95 y 99, lo que descarta la presencia de comportamientos erráticos o bimodales en el sistema durante la prueba.

- a) **Create:** mediana 1301 ms, $\rightarrow$ 90% 1983 ms, $\rightarrow$ 95% 2167 ms, $\rightarrow$ 99% 2501 ms

- b) **Read:** mediana 25 ms, → 90% 251 ms, → 95%, 319 ms, → 99% 498 ms
- c) **Update:** mediana 776 ms, → 90% 1402 ms, → 95% 1619 ms, → 99% 1911 ms
- d) **Delete:** mediana 1019 ms, → 90% 1591 ms, → 95% 1765 ms, → 99% 2076 ms

e) **Throughput similar entre operaciones**

El throughput se mantuvo prácticamente idéntico entre las cuatro operaciones, lo que evidencia un procesamiento predecible y balanceado del sistema bajo la carga aplicada, independientemente del tiempo de respuesta individual de cada operación.

- a) **Create:** 4.0 transacciones/seg
- b) **Read:** 4.0 transacciones/seg
- c) **Update:** 4.0 transacciones/seg
- d) **Delete:** 4.0 transacciones/seg

f) **Volumen y duración suficientes para diluir el ruido de corto plazo**

El volumen total de mediciones recolectadas durante los 30 minutos de prueba continua resulta suficiente para que efectos puntuales de corta duración.

- a) **Create:** 7561 mediciones
- b) **Read:** 7505 mediciones
- c) **Update:** 7502 mediciones
- d) **Delete:** 7470 mediciones

Total: 30038 mediciones en 30 minutos de prueba continua

4.6 **Análisis comparativo descriptivo entre los 2 aplicativos construidos bajo los paradigmas imperativo y reactivo**

Se realizará el análisis comparativo según las pruebas realizadas.

4.6.1 **Prueba de carga**

En el presente análisis se realiza la comparación de los resultados de las dos aplicaciones CRUD construidas. El objetivo es evaluar las diferencias en términos de rendimiento.

4.6.1.1 **Tiempo de Respuesta**

Con el propósito de realizar la comparación se realizó una tabla comparativa con los datos obtenidos de ambas aplicaciones donde se evaluó el tiempo de respuesta de cada prueba realizada carga, estrés y pico en las tablas 13, 15 y 17 respectivamente, en la cual muestra

los tiempos mínimo, máximo, promedio de respuesta de las operaciones, desv. Std., percentil 95, errores obtenidos y asimetría (Skewness) con el fin de identificar las diferencias encontradas entre ambos aplicativos. Estos indicadores se calcularon para cada endpoint con el propósito de no basarse en un único indicador: el promedio permite resumir un conjunto de datos en un solo valor; la desviación estándar mide cuánto se aleja cada dato del promedio; el coeficiente de variación mide la dispersión relativa al promedio; el P95 evalúa el tiempo bajo el cual se concentra el 95% de las peticiones; y la asimetría (Skewness) permite describir la forma de la distribución de los tiempos registrados.

Tabla 13

Tabla de datos de tiempo de respuesta para la prueba de carga

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	321.0	1288.71	3199.0	359.64	27.91%	1928.9	0%	0.667
	Read	5.0	405.81	2056.0	261.13	64.35%	862.0	0%	0.718
	Update	390.0	1787.86	3969.0	473.62	26.49%	2609.4	0%	0.33
	Delete	231.0	1054.62	2659.0	335.03	31.77%	1563.0	0%	0.648
Asíncrono	Create	336.0	1204.78	3111.0	323.71	26.87%	1792.85	0%	0.935
	Read	7.0	342.97	2404.0	178.52	52.05%	683.3	0%	1.896
	Update	195.0	1138.73	3831.0	404.96	35.56%	1854.0	0%	0.726
	Delete	251.0	1119.41	3066.0	345.92	30.9%	1766.0	0%	0.792

Nota. Elaboración propia.

Tabla 14

Tabla comparativa de tiempo de respuesta para la prueba de carga

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)

Create	83.93	6.51%	136.05	7.05%	1.04	0
Read	62.84	15.49%	178.70	20.73%	12.30	0
Update	649.13	36.31%	755.40	28.95%	-9.07	0
Delete	-64.79	6.14%	-113.0	6.84	0.87	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Tabla 15

Tabla de datos de tiempo de respuesta para la prueba de estrés

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	330.0	1147.97	3129.0	349.78	30.47%	1778.0	0	0.855
	Read	5.0	238.58	1717.0	265.75	111.39%	757.55	0	1.207
	Update	474.0	1507.0	4276.0	499.36	33.12%	2430.6	0	0.854
	Delete	272.0	924.0.	3008.0	336.57	36.39%	1547.0	0	1.328
Asíncrono	Create	310.0	1077.2	5544.0	297.85	27.65%	1606.0	0	1.8
	Read	8.0	265.09	2296.0	165.43	62.41%	556.7	0	2.091
	Update	167.0	843.0	4327.0	379.06	44.97%	1559.0	0	1.183

Delete	240.0	892.51	4321.0	311.39	34.89%	1484.0	0	1.175
---------------	-------	--------	--------	--------	--------	--------	---	-------

Nota. Elaboración propia.

Tabla 16

Tabla comparativa de tiempo de respuesta para la prueba de estrés

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)
Create	70.77	6.16%	172.00	9.67%	2.82	0
Read	-26.51	11.11%	200.85	26.51%	48.98	0
Update	664.59	44.08%	871.60	35.86%	11.85	0
Delete	32.28	3.49%	62.55	4.04%	1.50	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Tabla 17

Tabla de datos de tiempo de respuesta para la prueba de picos

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	322.0	1137.84	3161.0	340.57	29.93%	1795.65	0	0.953

Asíncrono	Read	5.0	259.18	1762.0	234.9	90.63%	705.0	0	1.314
	Update	472.0	1555.67	3904.0	477.5	30.69%	2460.4	0	0.924
	Delete	231.0	933.38	2746.0	336.16	36.02%	1571.65	0	1.306
	Create	322.0	1388.72	3335.0	417.16	30.04%	2167.0	0	0.719
	Read	8.0	77.39	821.0	110.8	143.16%	318.0	0	2.261
	Update	184.0	854.82	2564.0	365.37	42.74%	1619.0	0	1.025
	Delete	272.0	1089.72	2704.0	353.99	32.48%	1766.1	0	0.731

Nota. Elaboración propia.

Tabla 18

Tabla comparativa de tiempo de respuesta para la prueba de picos

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)
Create	-250.88	22.05%	370.35	20.61%	-0.11	0
Read	181.79	70.14%	386.20	54.78%	-52.53	0
Update	700.85	45.05%	841.40	34.2%	-12.05	0
Delete	156.34	16.75%	-194.45	12.37%	3.54	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Según los resultados obtenidos de las tablas 13,15 y 17 para:

a) Endpoint Create

Para el aplicativo asíncrono en la prueba de carga, presenta un promedio menor (1204.78 ms) frente a el aplicativo síncrono (1288.71 ms), con coeficientes de variación similares entre sí (26.87% y 27.91% respectivamente). Prueba de estrés, esta relación se mantiene, el aplicativo asíncrono (1077.2 ms) continúa siendo menor que el aplicativo síncrono (1147.97 ms), aunque su valor máximo se dispara hasta 5544.0 ms, mientras que su CV% se reduce ligeramente a 27.65%. Sin embargo, bajo la prueba de picos se produce una inversión, para el aplicativo síncrono desciende a 1137.84 ms, mientras que el aplicativo asíncrono se eleva a 1388.72 ms, con coeficientes de variación prácticamente idénticos entre ambos (29.93% y 30.04%). Este es el único endpoint donde el aplicativo con menor tiempo de respuesta promedio cambia según el tipo de prueba aplicado: el aplicativo asíncrono presentó menor tiempo de respuesta en los escenarios de carga y estrés, mientras que en el escenario de picos se invierte y es el aplicativo síncrono quien presenta el menor tiempo.

b) Endpoint Read

Este endpoint tiene una inestabilidad, en la prueba de carga, los CV% son elevados (64.35% aplicativo síncrono, 52.05% aplicativo asíncrono). En la prueba de estrés, el aplicativo síncrono se dispara a 111.39%, mientras que el aplicativo asíncrono se mantiene en 62.41%. En la prueba de picos, la dispersión alcanza una inestabilidad alta en ambos: 90.63% en el aplicativo síncrono y 143.16% en el aplicativo asíncrono. Cabe destacar que, bajo picos, el promedio del aplicativo asíncrono desciende a 77.39 ms, muy por debajo de los 259.18 ms del aplicativo síncrono. La inestabilidad se debe que la configuración de las pruebas se realizó teniendo como dependencia al anterior endpoint que es de create, primero debía ejecutarse y luego ejecutarse este endpoint es por eso que sale la inestabilidad.

c) Endpoint Update

Este es el endpoint en el que el aplicativo asíncrono reduce el tiempo promedio de forma sustancial frente a el aplicativo síncrono en los tres escenarios (1138.73 vs 1787.86 ms en carga; 843.0 vs 1507.0 ms en estrés; 854.82 vs 1555.67 ms en picos), siempre con un

CV% mayor al del aplicativo síncrono (35.56% vs 26.49% en carga; 44.97% vs 33.12% en estrés; 42.74% vs 30.69% en picos).

d) Endpoint Delete

En la prueba de carga, el aplicativo síncrono tiene un ligero tiempo de respuesta menor (1054.62 vs 1119.41 ms). En la prueba de estrés, la relación se invierte, el aplicativo asíncrono resulta tener el menor tiempo de respuesta (892.51 vs 924.0 ms), con un CV% también menor (34.89% vs 36.39%). En picos, la tendencia vuelve a invertirse a favor del aplicativo Síncrono (933.38 vs 1089.72 ms). Este comportamiento oscilante indica que, a diferencia de Update, en Delete ninguno de los dos aplicativos mantiene un tiempo menor fijo a través de los distintos escenarios de prueba.

Según los resultados obtenidos de las tablas 14,16 y 18 para:

a) Endpoint Create

En la prueba de carga, la diferencia de tiempo promedio entre ambos aplicativos fue de 83.93 ms (6.51%), con una diferencia de P95 de 136.05 ms (7.05%) y una diferencia de CV% de 1.04 puntos. En la prueba de estrés, la diferencia de promedio se redujo a 70.77 ms (6.16%), con una diferencia de P95 de 172.00 ms (9.67%) y una diferencia de CV% de 2.82 puntos. En la prueba de picos, el signo de la diferencia se invierte: el promedio registra una diferencia de -250.88 ms (22.05%) a favor del aplicativo sincrónico y el P95 de 370.35 ms (20.61%), mientras que la diferencia de CV% pasa a -0.11 puntos a favor del aplicativo sincrónico.

b) Endpoint Read

Bajo carga, la diferencia de promedio fue de 62.84 ms (15.49%) y la de P95 de 178.70 ms (20.73%), con una diferencia de CV% de 12.30 puntos. Bajo estrés, la diferencia de promedio cambia de signo (-26.51 ms, 11.11%), mientras que la diferencia de P95 se mantiene en el mismo sentido que en carga (200.85 ms, 26.51%); la diferencia de CV% se amplía a 48.98 puntos. Bajo picos, la diferencia de promedio es de 181.79 ms (70.14%) y la de P95 de 386.20 ms (54.78%), con una diferencia de CV% de -52.53 puntos, la magnitud más alta registrada entre las tres pruebas para este endpoint.

c) Endpoint Update

En carga, la diferencia de promedio fue de 649.13 ms (36.31%) y la de P95 de 755.40 ms (28.95%), con una diferencia de CV% de -9.07 puntos. En estrés, la diferencia de promedio aumenta a 664.59 ms (44.08%) y la de P95 a 871.60 ms (35.86%), con una diferencia de CV% de -11.85 puntos. En picos, la diferencia de promedio es de 700.85 ms (45.05%) y la de P95 de 841.40 ms (34.2%), con una diferencia de CV% de -12.05 puntos. En las tres condiciones, la diferencia de CV% mantiene el mismo signo (negativo).

d) Endpoint Delete

Bajo carga, la diferencia de promedio fue de -64.79 ms (6.14%) y la de P95 de -113.0 ms (6.84%), con una diferencia de CV% de 0.87 puntos. Bajo estrés, la diferencia de promedio cambia de signo (32.28 ms, 3.49%) y la de P95 también (62.55 ms, 4.04%), con una diferencia de CV% de 1.50 puntos. Bajo picos, la diferencia de promedio vuelve a cambiar de signo (156.34 ms, 16.75%) y la de P95 igualmente (-194.45 ms, 12.37%), con una diferencia de CV% de 3.54 puntos.

4.6.1.2 Consumo de Sistema (CPU y JVM)

Con el propósito de analizar el consumo de recurso del sistema para evaluar el rendimiento en operaciones de almacenamiento y recuperación de información. Durante las pruebas se analizó el CPU, memoria y JVM mientras la aplicación ejecutaba procesos de persistencia bajo los dos paradigmas. Estas tablas comparativas permiten identificar como cada modelo gestiona los recursos del sistema durante la ejecución de las operaciones CRUD, al final se realizaron una tabla de análisis de recursos y una tabla de comparación para cada prueba realizada carga, estrés y picos.

Tabla 19

Tabla de datos de tiempo de consumo de recursos para la prueba de carga

Métrica	Unidad	Síncrono			Asíncrono			Percentil 95		Desviación estándar		Coeficiente de variación	
		Min.	Prom.	Max.	Min.	Prom.	Max.	Síncrono	Asíncrono	Síncrono	Asíncrono	Síncrono	Asíncrono.
			.			.		0	.	0	.		
CPU – SQL server	%	N/A	9.04	11.3	N/A	10.5	13	10.64	12.64	0.0491	0.0594	0.5%	0.6%
Memoria – SQL server	MB	2300	2361	2381	2407	2489	2502	304	2501	63.9	13.4	2.7%	0.5%
CPU aplicativo	%	N/A	4.59	11	N/A	5.84	12.3	7.68	10.92	0.0465	0.055	1.0%	0.9%
Memoria aplicativo	MB	404	456	477	398	563	580	67	580	12.6	36.2	2.8%	6.4%
JVM Heap	MB	22	78.1	132	10.7	64.2	110	124	103	31.6	26.6	40.5%	41.4%
JVM Threads	Cantidad	22	34.1	38	11	25.9	28	37	28	4.81	3.05	14.1%	11.8%
JVM GC pause	Ms	N/A	14.6	76	N/A	5.26	12					0.0%	0.0%

Nota. Elaboración propia.

Tabla 20

Tabla comparativa de tiempo de consumo de recursos para la prueba de carga

Métrica	Prom. sincrónico	Prom. asincrónico	Máximo entre ambas	Normalizado sincrónico	Normalizado asincrónico
CPU (%) - SQL	9.04	10.50	10.50	0.861	1.000
Memoria contenedor (MB) - SQL	2361.00	2489.00	2489.00	0.949	1.000
CPU (%) aplicativo	4.59	5.84	5.84	0.786	1.000
Memoria contenedor (MB) aplicativo	456.00	563.00	563.00	0.810	1.000
JVM Heap (MB)	78.10	64.20	78.10	1.000	0.822
JVM Threads (n°)	34.10	25.90	34.10	1.000	0.760
JVM GC Pause (ms)	14.60	5.26	14.60	1.000	0.360

Nota. Valore normalizado = valor promedio/ valor máximo de ambos. Elaboración propia.

Tabla 21

Tabla de datos de tiempo de consumo de recursos para la prueba de estrés

Métrica	Unidad	Síncrono			Asíncrono			Percentil 95		Desviación estándar		Coeficiente de variación	
		Min	Prom	Max.	Min.	Prom	Max.	Síncron	Asíncrono	Síncron	Asíncrono	Síncrono	Asíncrono.
		.	.			.		0	.	0	.		
CPU – SQL server	%	N/A	6.21	9.45	N/A	8.67	13.3	8.72	12.56	0.0487	0.0681	0.8%	0.8%
Memoria – SQL server	MB	2246	2254	2260	2476	2484	2491	294	2491	53.1	5.5	2.4%	0.2%
CPU aplicativo	%	N/A	2.08	3.42	N/A	3.44	5.77	3.188	5.4	0.0203	0.0327	1.0%	1.0%
Memoria aplicativo	MB	531	536	542	581	594	596	87.9	596	11.5	2.95	2.1%	0.5%
JVM Heap	MB	31.6	100	173	29.6	72.1	109	167	105	41.9	22.6	41.9%	31.3%
JVM Threads	Cantidad	22	30.4	36	30	31	32	35.1	32	5.5	0.816	18.1%	2.6%
JVM GC pause	Ms	N/A	7.29	14	N/A	4.12	9					0.0%	0.0%

Nota. Elaboración propia.

Tabla 22

Tabla comparativa de tiempo de consumo de recursos para la prueba de estrés

Métrica	Prom. sincrónico	Prom. asincrónico	Máximo entre ambas	Normalizado sincrónico	Normalizado asincrónico
CPU (%) - SQL	6.21	8.67	8.67	0.716	1.000
Memoria contenedor (MB) - SQL	2254.00	2484.00	2484.00	0.907	1.000
CPU (%) aplicativo	2.08	3.44	3.44	0.605	1.000
Memoria contenedor (MB) aplicativo	536.00	594.00	594.00	0.902	1.000
JVM Heap (MB)	100.00	72.10	100.00	1.000	0.721
JVM Threads (n°)	30.40	31.00	31.00	0.981	1.000
JVM GC Pause (ms)	7.29	4.12	7.29	1.000	0.565

Nota. Valore normalizado = valor promedio/ valor máximo de ambos. Elaboración propia.

Tabla 23

Tabla de datos de tiempo de consumo de recursos para la prueba de picos

Métrica	Síncrono	Asíncrono	Percentil 95	Desviación estándar	Coefficiente de variación
----------------	-----------------	------------------	---------------------	----------------------------	----------------------------------

	Unidad	Min.	Prom.	Max.	Min.	Prom.	Max.	Síncrono	Asíncrono	Síncrono	Asíncrono	Síncrono	Asíncrono.
Cpu – SQL server	%	N/A	7.37	9.46	N/A	10.1	13.8	9	12.6	0.0333	0.0483	0.5%	0.5%
Memoria – SQL server	MB	2227	2233	2237	2487	2493	2504	288	2502	50	4.98	2.2%	0.2%
Cpu aplicativo	%	N/A	2.52	3.41	N/A	3.98	5.92	3.168	5.28	0.0136	0.0234	0.5%	0.6%
Memoria aplicativo	MB	533	541	550	606	607	608	87.4	608	11.7	0.39	2.2%	0.1%
JVM Heap	MB	34.5	104	175	40.8	76	110	170	107	40.8	20.1	39.2%	26.4%
Jvm Threads	Cantidad	34	35.9	37	32	32	32	37	32	0.54	0	1.5%	0.0%
Jvm GC pause	Ms	N/A	14.3	74	N/A	4.5	17					0.0%	0.0%

Nota. Elaboración propia.

Tabla 24

Tabla comparativa de tiempo de consumo de recursos para la prueba de picos

Métrica	Prom. sincrónico	Prom. asincrónico	Máximo entre ambas	Normalizado sincrónico	Normalizado asincrónico
CPU (%) SQL	7.37	10.10	10.10	0.730	1.000
Memoria contenedor (MB) - SQL	2233.00	2493.00	2493.00	0.896	1.000
CPU (%) aplicativo	2.52	3.98	3.98	0.633	1.000
Memoria contenedor (MB) aplicativo	541.00	607.00	607.00	0.891	1.000
JVM Heap (MB)	104.00	76.00	104.00	1.000	0.731
JVM Threads (n°)	35.90	32.00	35.90	1.000	0.891
JVM GC Pause (ms)	14.30	4.50	14.30	1.000	0.315

Nota. Valore normalizado = valor promedio/ valor máximo de ambos. Elaboración propia.

Según los resultados obtenidos de las tablas 19, 21 y 23 para:

a) CPU – SQL Server

En la prueba de carga, el consumo promedio de CPU del servidor SQL es de 9.04% para el aplicativo síncrono y 10.5% para el aplicativo asíncrono, con coeficientes de variación bajos y similares en ambos (0.5% y 0.6%), indicando estabilidad en el consumo. En estrés, ambos valores descienden (6.21% y 8.67% respectivamente) manteniendo la misma relación, y bajo picos se ubican en 7.37% y 10.1%. En las tres pruebas, el aplicativo asíncrono demanda más CPU del servidor de base de datos que el aplicativo síncrono, con una brecha de magnitud similar en las tres condiciones.

b) Memoria – SQL Server

En las tres pruebas, el aplicativo síncrono se mantiene en un rango de 2233–2361 MB, mientras que el aplicativo asíncrono se ubica sistemáticamente por encima, entre 2484–2489 MB. Los coeficientes de variación son considerablemente menores en el aplicativo asíncrono (0.2%–0.5%) frente al aplicativo síncrono (2.2%–2.7%), lo que indica que, aunque el aplicativo asíncrono consume más memoria de base de datos, su consumo presenta menor dispersión a lo largo del tiempo.

c) CPU – Aplicativo

En la prueba carga, el aplicativo síncrono consume en promedio 4.59% de CPU frente al 5.84% del aplicativo asíncrono. Bajo estrés, ambos valores se reducen (2.08% y 3.44%), y bajo picos se ubican en 2.52% y 3.98%. El aplicativo asíncrona demanda mayor procesamiento en el propio aplicativo en las tres condiciones evaluadas.

d) Memoria – Aplicativo

En la prueba carga, el aplicativo síncrono usa en promedio 456 MB frente a 563 MB del aplicativo asíncrono. Bajo estrés, ambos valores aumentan (536 MB y 594 MB), y bajo picos continúan en ascenso (541 MB y 607 MB). El coeficiente de variación del aplicativo asíncrono es menor en las tres pruebas (0.1%–6.4%) comparado con el aplicativo síncrono, y particularmente bajo estrés y picos la diferencia no se ve mucho (0.5% y 0.1% en asíncrono frente a 2.1% y 2.2% en síncrono), dando con un CV% menor en el aplicativo asíncrono respecto al aplicativo síncrono en dichas condiciones.

e) JVM Heap

El aplicativo síncrono presenta un mayor uso de heap en las tres pruebas (78.1 MB en carga, 100 MB en estrés, 104 MB en picos) frente al aplicativo asíncrono (64.2 MB, 72.1 MB y 76 MB respectivamente). Dando a entender que el aplicativo asíncrono consume menos JVM Heap.

f) JVM Threads

El aplicativo síncrono utiliza más hilos que el aplicativo asíncrono en las tres pruebas: 34.1 frente a 25.9 en carga, 30.4 frente a 31.0 en estrés, y 35.9 frente a 32.0 en picos. Se observa, en la prueba de picos, el CV% del aplicativo asíncrono llega a 0.0%, mantuvo un número de hilos constante, mientras que el aplicativo síncrono mostró mayor variabilidad (1.5%).

g) JVM GC Pause

En las tres pruebas, el aplicativo asíncrono presenta un tiempo de pausa por recolección de basura considerablemente menor que el aplicativo síncrono: 5.26 ms frente a 14.6 ms bajo carga, 4.12 ms frente a 7.29 ms bajo estrés, y 4.5 ms frente a 14.3 ms bajo picos. Indicando que el aplicativo asíncrono experimenta interrupciones más breves relacionadas con la gestión de memoria.

Según los resultados obtenidos de las tablas 20, 22 y 24 para:

a) CPU – SQL Server

En la prueba de carga, el aplicativo sincrónico presentó un promedio de 9.04% (valor normalizado: 0.861) y el aplicativo asincrónico un promedio de 10.50% (valor normalizado: 1.000). En estrés, el aplicativo sincrónico registró 6.21% (normalizado: 0.716) y el aplicativo asincrónico 8.67% (normalizado: 1.000). En picos, el aplicativo sincrónico presentó 7.37% (normalizado: 0.730) y el aplicativo asincrónico 10.10% (normalizado: 1.000).

b) Memoria – SQL Server

En carga, el aplicativo sincrónico registró un promedio de 2361.00 MB (normalizado: 0.949) y el aplicativo asincrónico 2489.00 MB (normalizado: 1.000). En estrés, el aplicativo sincrónico presentó 2254.00 MB (normalizado: 0.907) y el aplicativo asincrónico 2484.00 MB (normalizado: 1.000). En picos, el aplicativo sincrónico registró

2233.00 MB (normalizado: 0.896) y el aplicativo asincrónico 2493.00 MB (normalizado: 1.000).

c) CPU – Aplicativo

En carga, el aplicativo sincrónico presentó un promedio de 4.59% (normalizado: 0.786) y el aplicativo asincrónico 5.84% (normalizado: 1.000). En estrés, el aplicativo sincrónico registró 2.08% (normalizado: 0.605) y el aplicativo asincrónico 3.44% (normalizado: 1.000). En picos, el aplicativo sincrónico presentó 2.52% (normalizado: 0.633) y el aplicativo asincrónico 3.98% (normalizado: 1.000).

d) Memoria – Aplicativo

En carga, el aplicativo sincrónico registró un promedio de 456.00 MB (normalizado: 0.810) y el aplicativo asincrónico 563.00 MB (normalizado: 1.000). En estrés, el aplicativo sincrónico presentó 536.00 MB (normalizado: 0.902) y el aplicativo asincrónico 594.00 MB (normalizado: 1.000). En picos, el aplicativo sincrónico registró 541.00 MB (normalizado: 0.891) y el aplicativo asincrónico 607.00 MB (normalizado: 1.000).

e) JVM Heap

En carga, el aplicativo sincrónico presentó un promedio de 78.10 MB (normalizado: 1.000) y el aplicativo asincrónico 64.20 MB (normalizado: 0.822). En estrés, el aplicativo sincrónico registró 100.00 MB (normalizado: 1.000) y el aplicativo asincrónico 72.10 MB (normalizado: 0.721). En picos, el aplicativo sincrónico presentó 104.00 MB (normalizado: 1.000) y el aplicativo asincrónico 76.00 MB (normalizado: 0.731).

f) JVM Threads

En carga, el aplicativo sincrónico presentó un promedio de 34.10 hilos (normalizado: 1.000) y el aplicativo asincrónico 25.90 hilos (normalizado: 0.760). En estrés, el aplicativo sincrónico registró 30.40 hilos (normalizado: 0.981) y el aplicativo asincrónico 31.00 hilos (normalizado: 1.000) la única condición en la que el valor normalizado máximo corresponde al aplicativo asincrónico en esta métrica. En picos, el aplicativo sincrónico presentó 35.90 hilos (normalizado: 1.000) y el aplicativo asincrónico 32.00 hilos (normalizado: 0.891).

g) JVM GC Pause

En carga, el aplicativo sincrónico registró un promedio de 14.60 ms (normalizado: 1.000) y el aplicativo asincrónico 5.26 ms (normalizado: 0.360). En estrés, el aplicativo sincrónico presentó 7.29 ms (normalizado: 1.000) y el aplicativo asincrónico 4.12 ms (normalizado: 0.565). En picos, el aplicativo sincrónico registró 14.30 ms (normalizado: 1.000) y el aplicativo asincrónico 4.50 ms (normalizado: 0.315).

4.6.1.3 Latencia en operaciones

Con el propósito de realizar la comparación se realizó una tabla comparativa con los datos obtenidos de ambas aplicaciones donde se evaluó el tiempo de respuesta de cada prueba realizada carga, estrés y pico en las tablas 25, 27 y 29 respectivamente, en la cual muestra la latencia mínimo, máximo, promedio, desv. Std., percentil 95, errores obtenidos y asimetría (Skewness) con el fin de identificar las diferencias encontradas entre ambos aplicativos. Estos indicadores se calcularon para cada endpoint con el propósito de no basarse en un único indicador: el promedio permite resumir un conjunto de datos en un solo valor; la desviación estándar mide cuánto se aleja cada dato del promedio; el coeficiente de variación mide la dispersión relativa al promedio; el P95 evalúa el tiempo bajo el cual se concentra el 95% de las peticiones; y la asimetría (Skewness) permite describir la forma de la distribución de los tiempos registrados.

Tabla 25

Tabla de datos de latencia para la prueba de carga

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	321.0	1288.07	3199.0	359.6	27.92%	1927.8	0%	0.668
	Read	5.0	405.51	2056.0	261.1	64.39%	862.0	0%	0.718
	Update	390.0	1787.37	3968.0	473.63	26.5%	2607.8	0%	0.331
	Delete	0	0	0	0	0%	0	0%	0
Asíncrono	Create	336.0	1204.77	3111.0	323.69	26.87%	1792.85	0%	0.934
	Read	7.0	342.96	2404.0	178.52	52.05%	683.3	0%	1.896
	Update	195.0	1138.73	3831.0	404.96	35.56%	1854.0	0%	0.726
	Delete	0	0	0	0	0	0	0%	0

Nota. Elaboración propia.

Tabla 26

Tabla comparativa de Latencia para la prueba de carga

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)

Create	83.30	6.47%	134.95	7.0%	1.05	0
Read	62.55	15.43%	178.70	20.73%	12.34	0
Update	648.64	36.29%	753.80	28.91%	-9.06	0
Delete	0	0	0	0	0	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Tabla 27

Tabla de datos de latencia para la prueba de estrés

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	330.0	1147.64	3129.0	349.70	30.47%	1778.0	0	0.855
	Read	5.0	238.38	1717.0	265.69	111.46%	757.0	0	1.208
	Update	474.0	1507.32	4276.0	499.28	33.12%	2429.6	0	0.855
	Delete	0	0.	0	0	0%	0	0	0
Asíncrono	Create	310.0	1077.2	5544.0	297.85	27.65%	1606.0	0	1.8
	Read	8.0	265.08	2296.0	165.43	62.41%	556.7	0	2.091

Update	167.0	842.99	4327.0	379.06	44.97%	1559.0	0	1.183
Delete	0	0	0	0	0%	0	0	0

Nota. Elaboración propia.

Tabla 28

Tabla comparativa de Latencia para la prueba de estrés

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)
Create	70.44	6.14%	172.00	9.67%	2.82	0
Read	-26.70	11.2%	200.30	26.46%	49.05	0
Update	664.33	44.07%	870.60	35.83%	-11.85	0
Delete	0	0%	0	0%	0	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Tabla 29

Tabla de datos de latencia para la prueba de picos

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
-------------------	-----------------	------------	--------------	------------	-----------------------	------------	------------	---------------	-------------

Síncrono	Create	322.0	1137.45	3161.0	340.55	29.94%	1796.0	0	0.953
	Read	5.0	258.97	1761.0	234.85	90.69%	705.0	0	1.315
	Update	472.0	1555.36	3903.0	477.45	30.7%	2459.6	0	0.924
	Delete	0	0	0	0	0	0	0	0
Asíncrono	Create	322.0	1388.71	3335.0	417.16	30.04%	2167.0	0	0.719
	Read	8.0	77.39	821.0	110.8	143.18%	318.0	0	2.261
	Update	184.0	854.81	2564.0	365.37	42.74%	1619.0	0	1.025
	Delete	0	0	0	0	0	0	0	0

Nota. Elaboración propia.

Tabla 30

Tabla comparativa de Latencia para la prueba de picos

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)
Create	-251.26	22.09%	371.00	20.66%	-0.10	0
Read	181.58	70.12%	386.20	54.78%	-52.49	0
Update	700.55	45.04%	840.60	34.18%	-12.04	0
Delete	0	0	0	0	0	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Según los resultados obtenidos de las tablas 25,27 y 29 para:

a) Endpoint Create

En el endpoint Create, el aplicativo asíncrono se mantiene más rápido que el aplicativo síncrono bajo carga (1204.77 vs 1288.07 ms) y estrés (1077.2 vs 1147.64 ms), con niveles de estabilidad similares en ambos escenarios (CV% cercano al 27-28%). Bajo picos, sin embargo, la tendencia se invierte, el aplicativo síncrono baja a 1137.45 ms mientras el aplicativo asíncrono sube a 1388.71 ms, aunque el CV% se mantiene prácticamente igual entre ambos (30%).

b) Endpoint Read

Este endpoint (Read) es más inestable de la tabla. Bajo carga, el CV% ya es elevado (64.39% síncrono, 52.05% asíncrono). Bajo estrés, el aplicativo síncrono se dispara a 111.46%, mientras el aplicativo asíncrono se mantiene en 62.41%. Bajo picos se alcanza el extremo: 90.69% en Síncrono y 143.18% en asíncrono. Esto es debido a que las configuraciones de las pruebas ya que este endpoint dependencia exclusivamente del anterior que era create es por eso que la inestabilidad se notó.

c) Endpoint Update

En Update, el aplicativo asíncrono reduce el tiempo promedio, pero con mayor CV% que el aplicativo síncrono. Bajo carga, 26.5% vs 35.56%; bajo estrés: 33.12% vs 44.97%; bajo picos: 30.7% vs 42.74%. La diferencia de CV% se ubica entre 9 y 12 puntos porcentuales en las tres condiciones, lo que indica que la relación entre menor tiempo de respuesta y mayor dispersión relativa del aplicativo asíncrono se presenta en las tres pruebas evaluadas.

d) Endpoint Delete

En las tres pruebas carga, estrés y picos ambos aplicativos registran un valor de 0 en la totalidad de los indicadores de latencia. El endpoint responde a que esta operación retorna un código de respuesta sin cuerpo (204 No Content), por lo cual JMeter no logra registrar un valor de latencia diferenciado para esta operación, independientemente del escenario de prueba ejecutado.

Según los resultados obtenidos de las tablas 26, 28 y 30:

a) Endpoint Create

En carga, la diferencia de promedio fue de 83.30 ms (6.47%) y la de P95 de 134.95 ms (7.0%), con una diferencia de CV% de 1.05 puntos. En estrés, la diferencia de promedio fue de 70.44 ms (6.14%) y la de P95 de 172.00 ms (9.67%), con una diferencia de CV% de 2.82 puntos. En picos, el signo de la diferencia se invierte: el promedio registra -251.26 ms (22.09%) y el P95, 371.00 ms (20.66%), con una diferencia de CV% de -0.10 puntos.

b) Endpoint Read

Bajo carga, la diferencia de promedio fue de 62.55 ms (15.43%) y la de P95 de 178.70 ms (20.73%), con una diferencia de CV% de 12.34 puntos. Bajo estrés, la diferencia de promedio cambia de signo (-26.70 ms, 11.2%), mientras que la diferencia de P95 se mantiene en el mismo sentido (200.30 ms, 26.46%); la diferencia de CV% se amplía a 49.05 puntos. Bajo picos, la diferencia de promedio es de 181.58 ms (70.12%) y la de P95 de 386.20 ms (54.78%), con una diferencia de CV% de -52.49 puntos, la magnitud más alta registrada entre las tres pruebas para este endpoint.

c) Endpoint Update

En carga, la diferencia de promedio fue de 648.64 ms (36.29%) y la de P95 de 753.80 ms (28.91%), con una diferencia de CV% de -9.06 puntos. En estrés, la diferencia de promedio aumenta a 664.33 ms (44.07%) y la de P95 a 870.60 ms (35.83%), con una diferencia de CV% de -11.85 puntos. En picos, la diferencia de promedio es de 700.55 ms (45.04%) y la de P95 de 840.60 ms (34.18%), con una diferencia de CV% de -12.04 puntos. En las tres condiciones, la diferencia de CV% mantiene el mismo signo (negativo).

d) Endpoint Delete

En las tres pruebas (carga, estrés y picos), todos los indicadores de diferencia (Promedio, P95, CV%) registran valor 0, consecuencia de que ambos aplicativos presentan latencia constante de 0 ms en esta operación, dado que retorna un código de estado HTTP 204 (No Content), el cual no incluye cuerpo de respuesta.

4.6.1.4 Throughput

Con el propósito de realizar la comparación se realizó una tabla comparativa con los datos obtenidos de ambas aplicaciones donde se evaluó el tiempo de respuesta de cada prueba realizada carga, estrés y pico en las tablas 31, 33 y 35 respectivamente, en la cual muestra

el TPS mínimo, máximo, promedio, desv. Std., percentil 95, errores obtenidos y asimetría (Skewness) con el fin de identificar las diferencias encontradas entre ambos aplicativos. Estos indicadores se calcularon para cada endpoint con el propósito de no basarse en un único indicador: el promedio permite resumir un conjunto de datos en un solo valor; la desviación estándar mide cuánto se aleja cada dato del promedio; el coeficiente de variación mide la dispersión relativa al promedio; el P95 evalúa el tiempo bajo el cual se concentra el 95% de las peticiones; y la asimetría (Skewness) permite describir la forma de la distribución de los tiempos registrados.

Tabla 31

Tabla de datos de throughput para la prueba de carga

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	0	3.7	14	2.52	68.16%	0.8	0	0.602
	Read	0	3.7	14	2.41	65.18%	0.6	0	0.546
	Update	0	3.7	14	2.59	70.00%	0.6	0	0.574
	Delete	0	3.7	14	2.44	65.85%	3.85	0	0.5
Asíncrono	Create	0	4.41	13	2.45	55.5%	1.0	0	0.394
	Read	0	4.42	13	2.51	56.69%	1.0	0	0.396
	Update	0	4.42	13	2.55	57.71%	2.1	0	0.351
	Delete	0	4.42	14	2.51	59.15%	1	0	0.428

Nota. Elaboración propia.

Tabla 32

Tabla comparativa de throughput para la prueba de carga

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)

Create	-0.71	19.19%	-0.20	25.0%	12.66	0
Read	-0.72	19.46%	-0.40	66.67%	8.49	0
Update	-0.72	19.46%	-1.50	250.0%	12.29	0
Delete	-0.72	19.46%	2.85	74.03%	6.70	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Tabla 33

Tabla de datos de throughput para la prueba de estrés

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	0	2.82	14	2.74	97.05%	0	0	0.959
	Read	0	2.81	11	2.71	96.49%	1.75	0	0.864
	Update	0	2.81	13	2.82	100.37%	1.75	0	0.947
	Delete	0	2.8	12	2.76	98.42%	0	0	0.912
Asíncrono	Create	0	3.5	13	2.26	64.51%	2.0	0	0.893
	Read	0	3.49	13	2.28	65.28%	1.7	0	1.034
	Update	0	3.49	15	2.34	67.04%	2.3	0	1.102

Delete	0	3.48	14	2.33	66.86%	1.3	0	1.144
---------------	---	------	----	------	--------	-----	---	-------

Nota. Elaboración propia.

Tabla 34

Tabla comparativa de throughput para la prueba de estrés

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)
Create	-0.68	24.11%	-2.0	0%	32.54	0
Read	-0.68	24.2%	0.05	2.86%	31.21	0
Update	-0.68	24.2%	-0.55	31.43%	33.33	0
Delete	-0.68	24.29%	-1.3	0%	31.56	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Tabla 35

Tabla de datos de throughput para la prueba de picos

Aplicativo	Endpoint	Min	Prom.	Max	Desv. Std.	CV%	P95	Error%	Skew
Síncrono	Create	0	3.5	15	3.18	90.81%	2.9	0	1.188
	Read	0	3.48	15	3.2	91.98%	1.4	0	1.172

Asíncrono	Update	0	3.47	15	3.26	93.81%	2.9	0	1.145
	Delete	0	3.45	15	3.23	93.46%	3.0	0	1.208
	Create	0	3.98	12	2.2	55.14%	3.2	0	0.337
	Read	0	3.95	12	2.24	56.6%	6.4	0	0.303
	Update	0	3.95	13	2.29	57.94%	2.9	0	0.36
	Delete	0	3.93	11	2.29	58.13%	3.8	0	0.29

Nota. Elaboración propia.

Tabla 36

Tabla comparativa de throughput para la prueba de picos

Endpoint	Diferencias					
	Promedio	Promedio(%)	P95	P95(%)	CV(%)	Error(%)
Create	-0.48	13.71%	-0.30	10.34%	35.67	0
Read	-0.47	13.51%	-5.00	357.14%	35.38	0
Update	-0.48	13.83%	0	0%	35.87	0
Delete	-0.48	13.91%	-0.80	26.67%	35.33	0

Nota. Promedio = P. sin. – P. asin.; Promedio (%) = |Promedio| / P. sin. *100; P95 = p95_sin – p95_asin; P95 (%) = |P95| / p95_sin. P*100; CV%= CV% sin. – CV% asin. Elaboración propia.

Según los resultados obtenidos de las tablas 31, 33 y 35 para:

a) Endpoint Create

Bajo carga, el aplicativo asíncrono supera al aplicativo síncrono en throughput (4.41 vs 3.7 tx/s) y con menor CV% (55.5% vs 68.16%). Bajo estrés, ambos reducen su capacidad (2.82 vs 3.5 tx/s), pero la estabilidad se amplía, el CV% del aplicativo síncrono llega a 97.05%. Bajo picos, ambos se recuperan parcialmente (3.5 vs 3.98 tx/s), pero el aplicativo síncrono mantiene una dispersión muy alta (90.81% vs 55.14%). En las tres pruebas, el aplicativo asíncrono presentó mayor throughput promedio y menor CV% respecto al aplicativo síncrono

b) Endpoint Read

Para la prueba de carga, el aplicativo asíncrono (4.42 tx/s, CV% 56.69%) supera a el aplicativo síncrono (3.7 tx/s, CV% 65.18%). Bajo estrés, el aplicativo síncrono llega a un CV% de 96.49%, mientras el aplicativo asíncrono se mantiene en 65.28%. Bajo picos se registra el hallazgo, el P95 del aplicativo asíncrono asciende a 6.4 tx/s, muy por encima del 1.4 tx/s del aplicativo síncrono, lo que indica que, en los momentos de mayor throughput registrado, el aplicativo asincrónico alcanza valores considerablemente más altos que el aplicativo síncrono en esta operación.

c) Endpoint Update

Para la prueba de carga, el aplicativo asíncrono (4.42 tx/s, CV% 57.71%) vuelve a superar al aplicativo síncrono (3.7 tx/s, CV% 70.00%). Bajo estrés, la diferencia de promedio se mantiene similar (2.81 vs 3.49 tx/s), con síncrono alcanzando el CV% más alto de toda la tabla de estrés (100.37%). Bajo picos, ambos convergen a un P95 prácticamente idéntico (2.9 tx/s cada uno), aunque su CV% (93.81%) sigue siendo sustancialmente mayor al de el aplicativo asíncrono (57.94%).

d) Endpoint Delete

Para la prueba de carga, el P95 del aplicativo síncrono llega a 3.85 tx/s, muy por encima del 1.0 tx/s del aplicativo asíncrono, pese a que su promedio (3.7 tx/s) sigue siendo menor. Esto indica que, en los momentos de mayor throughput, el aplicativo síncrono tuvo picos de throughput más altos que el aplicativo asíncrono en esta operación. Bajo estrés, el aplicativo asíncrono (3.48 tx/s, CV% 66.86%) supera al aplicativo síncrono (2.8 tx/s,

CV% 98.42%). Bajo picos, la diferencia es moderada (3.45 vs 3.93 tx/s), con el aplicativo asíncrono presentando un CV% menor (58.13% vs 93.46%).

Según los resultados obtenidos de las tablas 32, 34 y 36 para:

a) Endpoint Create

En carga, la diferencia de promedio fue de -0.71 tx/s (19.19%) y la de P95 de -0.20 (25.0%), con una diferencia de CV% de 12.66 puntos. En estrés, la diferencia de promedio fue de -0.68 tx/s (24.11%) y la de P95 de -2.0 (0%), con una diferencia de CV% de 32.54 puntos. En picos, la diferencia de promedio fue de -0.48 tx/s (13.71%) y la de P95 de -0.30 (10.34%), con una diferencia de CV% de 35.67 puntos. En las tres condiciones, el signo de la diferencia de promedio se mantiene constante (negativo).

b) Endpoint Read

En carga, la diferencia de promedio fue de -0.72 tx/s (19.46%) y la de P95 de -0.40 (66.67%), con una diferencia de CV% de 8.49 puntos. En estrés, la diferencia de promedio fue de -0.68 tx/s (24.2%) y la de P95 de 0.05 (2.86%) cambiando de signo respecto a carga, con una diferencia de CV% de 31.21 puntos. En picos, la diferencia de promedio fue de -0.47 tx/s (13.51%) y la de P95 de -5.00 (357.14%), el valor porcentual más alto registrado en las tres tablas de Throughput, con una diferencia de CV% de 35.38 puntos.

c) Endpoint Update

En carga, la diferencia de promedio fue de -0.72 tx/s (19.46%) y la de P95 de -1.50 (250.0%), con una diferencia de CV% de 12.29 puntos. En estrés, la diferencia de promedio fue de -0.68 tx/s (24.2%) y la de P95 de -0.55 (31.43%), con una diferencia de CV% de 33.33 puntos. En picos, la diferencia de promedio fue de -0.48 tx/s (13.83%) y la de P95 de 0 (0%), con una diferencia de CV% de 35.87 puntos.

d) Endpoint Delete

En carga, la diferencia de promedio fue de -0.72 tx/s (19.46%) y la de P95 de 2.85 (74.03%) con signo distinto al de promedio, con una diferencia de CV% de 6.70 puntos, el valor más bajo registrado entre los cuatro endpoints en la prueba de carga. En estrés, la diferencia de promedio fue de -0.68 tx/s (24.29%) y la de P95 de -1.3 (0%), con una diferencia de CV% de 31.56 puntos. En picos, la diferencia de promedio fue de -0.48 tx/s (13.91%) y la de P95 de -0.80 (26.67%), con una diferencia de CV% de 35.33 puntos.

4.7 Interpretación de resultados

Los resultados obtenidos permiten sustentar, con evidencia empírica, los fundamentos teóricos que diferencian entre los aplicativos construidos bajo el paradigma reactivo del paradigma imperativo. El aplicativo construido bajo el paradigma reactivo, al operar bajo un esquema de entrada/salida no bloqueante basado en un bucle de eventos, no reserva un hilo de ejecución exclusivo por cada solicitud entrante, sino que libera el hilo tan pronto la operación queda a la espera de una respuesta externa, una consulta a base de datos, permitiéndole atender otras solicitudes mientras dicha operación se completa. Esta característica explica de manera directa el mayor throughput que el aplicativo asíncrono presenta a lo largo de las tres condiciones de prueba evaluadas: al no depender de la disponibilidad de un hilo dedicado por solicitud, su capacidad de procesamiento concurrente se ve menos condicionada por el número de peticiones simultáneas que enfrenta el sistema.

El aplicativo construido bajo el paradigma imperativo, por el contrario, opera bajo un esquema de ejecución bloqueante en el cual cada solicitud ocupa un hilo del pool de manera continua hasta que su procesamiento finaliza por completo. Este comportamiento se ve reflejado en el incremento progresivo del coeficiente de variación del throughput, pasando de niveles moderados bajo carga a valores cercanos al 100%, prácticamente igualando en magnitud a su propio promedio. Este comportamiento es notable con la saturación del pool de hilos: a medida que aumenta la concurrencia, un número creciente de solicitudes debe permanecer en espera de que un hilo quede disponible, generando una variabilidad considerablemente mayor en la capacidad de respuesta del sistema.

La mejora del asíncrono resultó particularmente pronunciada en la operación de actualización (Update), donde se registró la mayor diferencia porcentual de tiempo de respuesta. Este comportamiento es coherente con el hecho de que las operaciones de escritura suelen requerir un mayor tiempo de procesamiento a nivel de base de datos, escenario en el cual el modelo no bloqueante obtiene mayor provecho al liberar el hilo de ejecución mientras la operación de persistencia se completa de forma asíncrona. No obstante, el coeficiente de variación del aplicativo asíncrono resultó sistemáticamente mayor al del síncrono en esta operación. Esto sugiere que, si bien el síncrono logra reducir el tiempo de procesamiento en promedio, la naturaleza de la programación de tareas dentro del bucle de eventos depende de la disponibilidad del hilo.

Por otra parte, la operación de lectura (Read) se notó un comportamiento más inestable de las cuatro evaluadas en ambos aplicativos, con coeficientes de variación que fueron altos. Este patrón no es atribuible a ni uno de los aplicativos desarrollados ya que fue parte de la configuración de las pruebas de carga, picos y estrés. Ya que para que se ejecute la operación se tuvo primero que ejecutar el anterior endpoint que fue de la creación, estaba condicionado a la respuesta de la creación es por eso que varía demasiado en el coeficiente de variación.

Un hallazgo que se observó fue en la operación de creación (Create) bajo la prueba de picos, escenario en el cual el aplicativo síncrono tiene menor tiempo de respuesta comparado con el de asíncrono, invirtiéndose los resultados de las pruebas de carga y estrés. Esto puede explicarse, mientras que el modelo de hilos del aplicativo síncrono cuenta con un pool de hilos ya aprovisionado capaz de absorber, hasta cierto límite, una ráfaga de solicitudes de forma inmediata, el aplicativo asíncrono puede experimentar una saturación puntual del bucle de eventos cuando la ráfaga de solicitudes concurrentes supera su capacidad, generando una acumulación temporal de tareas pendientes de ejecución. Este comportamiento sugiere que el menor tiempo de respuesta del aplicativo asincrónico se maximiza en escenarios de carga sostenida, mientras que dicha diferencia podría atenuarse o incluso revertirse ante incrementos abruptos y de corta duración en el volumen de solicitudes.

Respecto al consumo de recursos, los resultados evidencian que el aplicativo asincrónico no se manifiesta en un menor uso de recursos de infraestructura. Por el contrario, el aplicativo asincrónico demandó mayor porcentaje de CPU y mayor cantidad de memoria, tanto a nivel de base de datos como del propio aplicativo, en las tres condiciones de prueba. Este comportamiento resulta coherente con la sobrecarga computacional inherente a la gestión del bucle de eventos y a la programación de rutinas, si bien evita el bloqueo de hilos, introduce un costo de procesamiento adicional asociado a la administración y coordinación de las tareas concurrentes en ejecución.

En contraste, el comportamiento de la máquina virtual Java mostró una tendencia inversa: el aplicativo asincrónico presentó un menor uso de memoria heap, un menor número de hilos activos y, de forma particularmente marcada, tiempos de pausa por recolección de basura considerablemente más breves que el aplicativo síncrono en las tres pruebas evaluadas. Esta diferencia puede atribuirse a que el modelo no bloqueante, al no mantener hilos en espera de operaciones de entrada/salida, genera un menor número de objetos

retenidos en memoria durante los periodos de espera, reduciendo la presión sobre el recolector de basura y, en consecuencia, disminuyendo tanto la frecuencia como la duración de sus pausas. Esta evidencia resulta relevante, dado que un menor tiempo de pausa por recolección de basura se traduce directamente en una mayor previsibilidad del tiempo de respuesta bajo cargas sostenidas.

En conjunto, estos resultados permiten identificar diferencias de comportamiento entre ambos aplicativos que no deben interpretarse como que uno de los dos aplicativos presenta mejores resultados favorables en la totalidad de las métricas evaluadas: el aplicativo asincrónico presenta menor consumo de hilos y de memoria a nivel de la máquina virtual, a cambio de una mayor demanda de procesamiento y memoria a nivel de infraestructura, así como una mayor variabilidad ante incrementos abruptos de tráfico en determinadas operaciones

4.8 Contrastación descriptiva de hipótesis

En base a los resultados obtenidos a partir del análisis descriptivo (promedio, valores máximos y mínimos. Desviación estándar, p95, CV%, Skew) de las métricas evaluadas, se procede a la contrastación descriptiva de las hipótesis planteadas:

- a) Hipótesis específica 1, los resultados evidenciaron diferencias en el tiempo de respuesta promedio entre los aplicativos desarrollados bajo ambos paradigmas en los cuatro endpoints evaluados y bajo las tres condiciones de prueba. El aplicativo asincrónico registró menores tiempos promedio en Create, Read y Update, mientras que el aplicativo sincrónico registró menores tiempos promedio en Delete y, puntualmente, en Create bajo la prueba de picos. Esto refleja la existencia de diferencias a nivel descriptivo entre ambos aplicativos desarrollados bajo ambos paradigmas, aunque no de forma absoluta en todos los escenarios.
- b) Hipótesis específica 2, se identificaron diferencias en el consumo de recursos entre ambos aplicativos a lo largo de las tres pruebas. El aplicativo asincrónico demandó mayor porcentaje de CPU y memoria tanto en el servidor de base de datos como en el propio aplicativo (normalizado en 1.000 en las tres pruebas), mientras que el aplicativo sincrónico mostró mayor consumo de memoria Heap, mayor número de hilos JVM y tiempos de pausa por recolección de basura considerablemente más altos. Estas diferencias muestran que ambos aplicativos

gestionan los recursos del sistema de manera distinta, aunque en direcciones opuestas según el tipo de recurso evaluado.

- c) Hipótesis específica 3, la latencia mostró un comportamiento equivalente al tiempo de respuesta en las tres pruebas, evidenciando diferencias entre ambos aplicativos en tres de los cuatro endpoints (Create, Read y Update), con el aplicativo asincrónico presentando menor latencia. La operación Delete no permitió establecer diferencias, al registrar valores nulos en ambos aplicativos debido a la naturaleza de la respuesta HTTP (204 No Content).
- d) Hipótesis específica 4, el throughput fue la métrica donde se observó la diferencia más gradual entre ambos aplicativos: el aplicativo asincrónico obtuvo mayor TPS con respecto al aplicativo sincrónico en throughput promedio en prácticamente la totalidad de las combinaciones de endpoint y prueba evaluadas además de mostrar un coeficiente de variación considerablemente menor, bajo las distintas condiciones de carga aplicadas.

Las cuatro hipótesis de investigación fueron aceptadas, al haberse identificado diferencias a nivel descriptivo entre los aplicativos desarrollados bajo los paradigmas imperativo y reactivo en las cuatro dimensiones evaluadas (tiempo de respuesta, consumo de recursos, latencia y throughput). No obstante, dichas diferencias no favorecen de manera absoluta a un único aplicativo, sino que revelan un patrón: el aplicativo asincrónico presenta, en la mayoría de los escenarios evaluados, menores valores en tiempo de respuesta y latencia, y mayores valores de throughput, mientras que el aplicativo sincrónico presenta, en contraparte, menor demanda de recursos de infraestructura (CPU y memoria a nivel de base de datos y aplicativo)

4.9 Discusión del resultado

En esta sección se analizan e interpretan los resultados obtenidos durante la ejecución de las pruebas de carga, estrés y picos aplicadas a las aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo, contrastando los hallazgos con la evidencia reportada en investigaciones previas.

Con respecto con el tiempo de respuesta, los resultados evidencian que el aplicativo asincrónico mantuvo tiempos de respuesta promedio menores frente al aplicativo sincrónico en la mayoría de las operaciones y escenarios evaluados. Durante las pruebas de carga y estrés, el aplicativo asincrónico procesó las solicitudes de Create, Read y

Update en menor tiempo, alcanzando en la operación Update la diferencia más vista. Sin embargo, este comportamiento no se sostuvo de manera absoluta, en la operación Delete el modelo sincrónico presentó menores tiempos en dos de las tres pruebas, en la operación Create bajo la prueba de picos se invirtió, resultando el modelo sincrónico con tiempos menores. Esta inversión sugiere que el aplicativo asincrónico se maximiza ante un incremento gradual y sostenido de solicitudes, mientras que podría verse comprometida ante súbitas de tráfico que saturan momentáneamente el bucle de eventos. Estos resultados coinciden parcialmente con lo reportado por Vallejo Fernández (2022), cuyo estudio también documentó menores tiempos promedio de respuesta en el modelo asincrónico, aunque sin evidencia en el comportamiento de picos o incrementos abruptos de carga.

En cuanto a consumo de CPU, los resultados muestran que el aplicativo asincrónico presentó un mayor consumo de CPU respecto al aplicativo sincrónico, tanto a nivel del servidor de base de datos como del propio aplicativo, en las tres condiciones de prueba evaluadas. Este resultado coincide con lo señalado por Iwanowski y Kozieł (2022) y por Karol Lis y Jakub Smolka (2024), quienes también reportaron un mayor uso de CPU en el modelo asincrónico respecto al sincrónico.

Por otro lado, el consumo de memoria JVM, el comportamiento de la memoria gestionada por la máquina virtual Java mostró un patrón distinto al de la CPU. Si bien ambos aplicativos operaron dentro de rangos relativamente cercanos, el aplicativo sincrónico presentó un mayor uso de memoria Heap y un mayor número de hilos activos en las tres pruebas realizadas. Este hallazgo complementa lo señalado por Karol Lis y Jakub Smolka (2024): mientras que el modelo reactivo demanda mayor CPU para la coordinación de sus tareas concurrentes, dicha sobrecarga de procesamiento no se traduce en una mayor presión sobre la memoria interna de la JVM, sino en un uso de menor magnitud de esta.

Con respecto a la latencia, se evidencia un comportamiento equivalente al observado en el tiempo de respuesta, evidenciando que el aplicativo asincrónico presentó menores tiempos en las operaciones de escritura, particularmente en Create y Update, donde los tiempos promedio resultaron menores en las tres pruebas realizadas. En las operaciones de lectura (Read), si bien el aplicativo asincrónico también registró menores promedios, ambos aplicativos mostraron niveles de dispersión altos, con coeficientes de variación superiores al 50% en la mayoría de los escenarios, esto debido a la configuración de las pruebas ya que dependencia mucho de la anterior operación que era de create. En la

operación Delete, la latencia no permitió establecer diferencias, al registrarse valores nulos en ambos aplicativos como consecuencia de la naturaleza de la respuesta HTTP sin contenido devuelta por esta operación. Estos hallazgos coinciden con lo reportado por Vallejo Fernández (2022), quien documentó menores tiempos de latencia en el modelo asincrónico para las operaciones de escritura.

Finalmente, el throughput fue la métrica que evidenció la diferencia más notable entre ambos aplicativos. El aplicativo asincrónico obtuvo mayor TPS con respecto al aplicativo sincrónico en throughput promedio en prácticamente la totalidad de las combinaciones de endpoint y prueba evaluadas, el aplicativo asincrónico mantuvo sus dispersiones menores en los tres escenarios. Este resultado coincide lo señalado por Iwanowski y Kozieł (2022), quienes reportaron que el modelo asincrónico es capaz de procesar un mayor volumen de solicitudes por unidad de tiempo respecto al modelo sincrónico, y sugiere adicionalmente que dicha diferencia se acentúa no solo en magnitud, sino también a medida que se incrementa el nivel de concurrencia al que se somete el sistema.

Capítulo V

Conclusiones y recomendaciones

5 Conclusiones y recomendaciones

5.1 Conclusiones

- a. Se realizó la comparación del rendimiento en persistencia de datos de ambas aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativa y reactiva, cumpliendo con el objetivo principal de evaluarlos mediante las métricas del tiempo de respuesta, consumo de recurso de CPU Y JVM, throughput, y latencia bajo distintos escenarios de pruebas de carga, estrés y picos.
- b. Se cumplió el objetivo de comparar el tiempo de respuesta en operaciones CRUD entre aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo. Tal como se muestra en el Capítulo 4, los resultados obtenidos a partir de las pruebas permitieron comparar el comportamiento de ambas aplicaciones en términos de tiempo de respuesta durante la ejecución de las pruebas de carga como se muestra en la tabla 13 y 14, estrés en la tabla 15 y 16 y picos en la tabla 17 y 18, en las operaciones Create, Read y Update, el aplicativo desarrollado bajo el paradigma reactivo registró un menor tiempo de respuesta promedio respecto al aplicativo imperativo en las pruebas de carga y estrés; en la operación Delete, el aplicativo imperativo registró un menor tiempo de respuesta promedio en dos de las tres pruebas. Bajo la prueba de picos, la diferencia se invirtió en el endpoint Create, registrando el aplicativo imperativo un menor tiempo de respuesta promedio respecto al aplicativo reactivo. La operación Update presentó las diferencias más amplias entre ambos aplicativos en las tres pruebas, mientras que la operación Delete registró las diferencias más reducidas.
- c. Se cumplió el objetivo de evaluar el consumo de recursos del sistema, específicamente CPU y JVM, en las aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo durante operaciones de persistencia de datos. Tal como se muestra en el Capítulo 4, los resultados permitieron analizar y comparar el consumo de recursos de cada uno de los aplicativos en las pruebas de rendimiento que incluyeron escenarios de carga como se muestra en la tabla 19 y 20, estrés como se muestra en la tabla 21 y 22 y picos como se muestra en la tabla 23 y 24. En las métricas de CPU y memoria a nivel de base de datos y aplicativo,

el aplicativo reactivo presentó el valor normalizado máximo en las tres condiciones de prueba. En las métricas de la máquina virtual Java (JVM Heap, JVM Threads y JVM GC Pause), el aplicativo imperativo presentó el valor normalizado máximo, con excepción de JVM Threads bajo la prueba de estrés, donde el valor normalizado máximo correspondió al aplicativo reactivo. La métrica de JVM GC Pause registró la mayor diferencia entre los valores normalizados de ambos aplicativos, alcanzando el aplicativo reactivo un valor normalizado de 0.315 bajo la prueba de picos.

- d. Se cumplió el objetivo de comparar y evaluar la latencia en las operaciones CRUD de los aplicativos CRUD desarrolladas bajo los paradigmas imperativo y reactivo. Tal como se muestra en el Capítulo 4, los resultados obtenidos a partir de las pruebas de carga como se muestra en la tabla 25 y 26, estrés como se muestra en la tabla 27 y 28, y picos como se muestra en la tabla 29 y 30. Se identificó diferencias entre ambos aplicativos en tres de los cuatro endpoints evaluados (Create, Read y Update), con un comportamiento equivalente al observado en la métrica de tiempo de respuesta. En la operación Delete, no fue posible establecer diferencias entre ambos aplicativos, dado que la latencia se registró en 0 ms en las tres pruebas, como consecuencia de que dicha operación retorna un código de estado HTTP 204 (No Content), el cual no incluye cuerpo de respuesta.
- e. Se cumplió el objetivo de medir y comparar el throughput alcanzado por cada CRUD desarrolladas bajo los paradigmas imperativo y reactivo durante pruebas de rendimiento. Tal como se muestra en el Capítulo 4, los resultados obtenidos de la prueba de carga como se muestra en la tabla 31 y 32, prueba de estrés como se muestra en la tabla 33 y 34 y prueba de picos como se muestra en la tabla 35 y 36 obtenidos permitieron analizar, comparar e identificando diferencias en el volumen de transacciones procesadas durante los distintos escenarios de prueba. A diferencia de lo observado en tiempo de respuesta y latencia, en throughput el signo de la diferencia de promedio fue el mismo en los cuatro endpoints y en las tres condiciones de prueba, registrando el aplicativo reactivo un mayor throughput promedio respecto al aplicativo imperativo.

5.2 Recomendaciones

- a. Se recomienda ampliar el estudio con otras futuras investigaciones aumentando otros componentes como sistemas de cache Redis, mensajería de colas como Kafka y una interfaz. Esto permitirá realizar nuevas comparaciones de rendimiento en la persistencia de datos.
- b. Explorar una futura incorporación de balanceadores de carga y realizar diferentes pruebas de performance. Esto para analizar y medir con mayor precisión el rendimiento en alta concurrencia, lo que permitirá evaluar el impacto en el proceso de persistencia de datos en alta concurrencia.
- c. Se recomienda evaluar distintos motores de base de datos, tanto relaciones como no relacionales, con el fin de comparar el comportamiento de cada paradigma en rendimiento de persistencia de datos.
- d. Se recomienda realizarlo en entorno cloud para evaluar el costo computacional y operativo de cada paradigma asociado a las consultas y operaciones realizadas sobre la base de datos. Esto permitirá medir con mayor precisión el impacto que tiene los modelos sincrónicos y asincrónicos en el consumo de recursos en la base de datos.
- e. Se sugiere extender la investigación con otros lenguajes de programación y framework y diferentes bases de datos como relaciones y no relaciones. Esto permitirá comparar el comportamiento de los paradigmas evaluados en diferentes entornos tecnológicos y validar así los resultados obtenidos en relación con el rendimiento en persistencia de datos.
- f. Se recomienda que futuras investigaciones incluyan los Virtual Threads de Java, para manejar concurrencia sin usar programación reactiva.

Referencias bibliográficas

1. Aguilar Encarnación, P. S., Castillo, J. P., Carreño, H. J., & Jativa Brito, M. E. (2024). *Estudio comparativo de los paradigmas de programación orientada a objetos y programación reactiva en la resolución de Integrales Algebraicas*. Revista Tecnológica - ESPOL, 36(1), 58-67. <https://doi.org/10.37815/rte.v36n1.1163>
2. Cocca, A. (2022, 2 de mayo). *Programming Paradigms*. FreeCodeCamp. <https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/>
3. Apache Software Foundation. (2024). *JMeter Documentation*. <https://jmeter.apache.org>
4. Liyan, A. (2023). *What is a programming paradigm?* Medium. <https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b>
5. Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, J. C. (2004). *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE Transactions on Dependable and Secure Computing, 1(1), 11-33.
6. Bernal, C. A. (2010). *Metodología de la investigación: para administración, economía, humanidades y ciencias sociales* (3ª ed.). Pearson Educación.
7. Built In. (2025, 19 de febrero). *What is Imperative Programming? (Definition, Example)*. BuiltIn. <https://builtin.com/articles/imperative-programming>
8. Calzado Pérez, Á. (2020). *API REST en AWS para la estructuración y persistencia de datos obtenidos de WeSound y sistema de visualización basado en NODE.JS* [Tesis de pregrado, Universidad Politécnica de Madrid]. Archivo Digital UPM. <https://oa.upm.es/63459/>

9. Chapman, R., & Holmes, P. (2023). *Observability with Grafana*. Packt Publishing.
10. Coronel, C., & Morris, S. (2018). *Database Systems: Design, Implementation, & Management* (13th ed.). Cengage Learning.
11. Date, C. J. (2003). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.
12. Docker, Inc. (2024). What is Docker? Docker Docs. <https://docs.docker.com/>
13. Espinoza Rojas, J. (2022). *Comparativa del paradigma de programación reactiva frente al paradigma síncrono* [Tesis de pregrado, Universidad Politécnica de Madrid]. Archivo Digital UPM. <https://oa.upm.es/68027/>
14. Elradi, M. D. (2025). *Prometheus & Grafana: A metrics-focused monitoring stack*. *Journal of Computer Allied Intelligence*, 3(3), 28–39. <https://doi.org/10.69996/jcai.2025015>
15. Frey, S. (2022, 23 de mayo). *A comprehensive introduction to reactive programming*. LogRocket Blog. <https://blog.logrocket.com/comprehensive-introduction-reactive-programming/>
16. Koksai, G. (2024). *Evaluating the Effectiveness of Reactive Programming Compared to Traditional Methods for Enhancing Responsiveness, Resource Efficiency, and Scalability in Spring Boot Microservices*. ResearchGate. https://www.researchgate.net/publication/385680753_Evaluating_the_Effectiveness_of_Reactive_Programming_Compared_to_Traditional_Methods_for_Enhancing_Responsiveness_Resource_Efficiency_and_Scalability_in_Spring_Boot_Microservices
17. Hidalgo de Benito, C. (2021). *Monitorización de un entorno empresarial con Prometheus-Grafana* [Trabajo de fin de grado]. Escuela Técnica Superior de

- Ingenieros Informáticos, Universidad Politécnica de Madrid.
<https://oa.upm.es/68055/>
18. IBM Developer. (2025). *What is a REST API?*
<https://www.ibm.com/think/topics/rest-apis>
 19. IONOS. (2021, 21 de julio). *Programación imperativa: Ventajas y desventajas del paradigma*. IONOS. <https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/programacion-imperativa/>
 20. Iwanowski, S. & Kozieł, G. (2022). *Comparative analysis of reactive and imperative approach in Java web application development*. ResearchGate
https://www.researchgate.net/publication/364075699_Comparative_analysis_of_reactive_and_imperative_approach_in_Java_web_application_development
 21. Jader, M., Silva, R., et al. (2019). *Performance Efficiency Evaluation based on ISO/IEC 25010:2011 applied to a Case Study on Load Balance and Resilient*. ResearchGate.
https://www.researchgate.net/publication/371764897_Performance_Efficiency_Evaluation_based_on_ISOIEC_250102011_applied_to_a_Case_Study_on_Load_Balance_and_Resilient
 22. JetBrains. (s.f.). *IntelliJ IDEA: The Capable & Ergonomic Java IDE*.
<https://www.jetbrains.com/idea/>
 23. Jhontona. (2025). *Domina el presente y el futuro: Imperativo vs. reactivo, ¿cuál te conviene?* <https://jhontona.com/domina-el-presente-y-el-futuro-imperativo-vs-reactivo-cual-te-conviene/>
 24. Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2.

25. Nirmal, K. (2025, 16 de enero). *Reactive Programming: A Guide*.
<https://builtin.com/articles/reactive-programming>
26. Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media.
27. Krishnan, R. (2024, 30 de octubre). *Reactive Programming in Java: A Simplified Guide*. Medium. <https://solutionsarchitecture.medium.com/reactive-programming-in-java-a-simplified-guide-dd24d7f440d4>
28. Kurose, J. F., & Ross, K. W. (2017). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.
29. Lis, K., & Smółka, J. (2024). *Examination of the performance and scalability of a web application in a reactive and imperative approach using the Spring Framework*. *Journal of Computer Sciences Institute*. ResearchGate.
https://www.researchgate.net/publication/384478968_Examination_of_the_performance_and_scalability_of_a_web_application_in_a_reactive_and_imperative_approach_using_the_Spring_Framework
30. LoadFocus. (2013, julio). *What is Response Time in Performance Testing?* LoadFocus Blog. <https://loadfocus.com/blog/2013/07/what-is-response-time-in-performance-testing>
31. López, F., & Rojas, C. (2022). *Pruebas de rendimiento con JMeter: Una aproximación práctica*. *Revista de Ingeniería de Software*, 28(1), 45–53.
32. Microsoft. (2024). *¿Qué es SQL Server?* <https://learn.microsoft.com/es-es/sql/sql-server/>
33. Mochnej, K. & Badurowicz, M. (2023). *Performance comparison of microservices written using reactive and imperative approaches*. *Journal of Computer Sciences Institute*.

- https://www.researchgate.net/publication/374348603_Performance_comparison_of_microservices_written_using_reactive_and_imperative_approaches/citation/download
34. Nielsen, J. (1994, 1 de enero). *Response Times: The 3 Important Limits*. Nielsen Norman Group. NNGROUP. <https://www.nngroup.com/articles/response-times-3-important-limits/>
35. Pardo Calvache, C. J. (2009) *Análisis comparativo del estándar ISO 9000-3 con las subcaracterísticas de calidad de la ISO 9126* [Tecnologías Informáticas Avanzadas].
- https://www.researchgate.net/publication/341277269_Analisis_comparativo_del_estandar_ISO_9000-3_con_las_subcaracteristicas_de_calidad_de_la_ISO_9126
36. Pérez, Y. P., & López, L. M. (s.f.). *Multiparadigma en la enseñanza de la programación*. Universidad Nacional del Comahue. https://sedici.unlp.edu.ar/bitstream/handle/10915/20499/Documento_completo.pdf?sequence=1
37. Quitaran Perez, D. (2024, 30 de enero). *Programación Imperativa: Un Enfoque Paso a Paso para Resolver Problemas de Software*. LinkedIn. <https://es.linkedin.com/pulse/programaci%C3%B3n-imperativa-un-enfoque-paso-para-resolver-david-gdrte>
38. Red Hat. (2020). *What is a REST API?* <https://www.redhat.com/en/topics/api/what-is-a-rest-api>
39. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.

40. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education.
41. Skinait. (2023, 26 de julio). *Estilos de programación, enfoques y paradigmas*. <https://www.skinait.com/estilos-de-programaci%C3%B3n-enfoques-y-paradigmas-Escritos-78/>
42. Spring. (s.f.). *Spring Framework Documentation*. <https://docs.spring.io/spring-framework/reference/index.html>
43. Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson.
44. Vallejo Fernández, L. (2022). *Comparativa del paradigma de programación reactiva frente al paradigma síncrono* [Tesis de grado]. Universidad Politécnica de Madrid. <https://oa.upm.es/68027/>
45. Vara-Horna, Arístides (2010). *¿Cómo hacer una tesis en ciencias empresariales? Manual breve para los tesisas de Administración, Negocios Internacionales, Recursos Humanos y Marketing*. Facultad de Ciencias Administrativas y Recursos Humanos de la Universidad de San Martín de Porres. Lima – Perú. Segunda edición
46. Walls, C. (2018). *Spring in Action* (5th ed.). Manning Publications.
47. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2024). *Experimentation in software engineering* (2nd ed.). Springer.

Anexos

Anexo A: Matriz de consistencia

Análisis comparativo del rendimiento en persistencia de datos entre paradigmas imperativo y reactivo, mediante aplicaciones CRUD, Ayacucho 2025.

PROBLEMA	OBJETIVO	HIPÓTESIS	METODOLOGIA
Problema general ¿Cuáles son las diferencias en el rendimiento de la persistencia de datos entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en un entorno Java? Problemas específicos a. ¿Qué diferencias existen en el tiempo de respuesta entre aplicaciones CRUD desarrolladas bajo los paradigmas imperativo y reactivo durante las	Objetivo general Comparar el rendimiento en persistencia de datos en las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo, mediante escenarios de pruebas de carga, estrés y picos. Objetivos específicos a) Comparar el tiempo de respuesta de las aplicaciones CRUD desarrolladas bajo los paradigmas de	Hipótesis general Existen diferencias en el rendimiento de la persistencia de datos entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo, en los escenarios de pruebas de carga, estrés y picos. Hipótesis específicas a) Existen diferencias en el tiempo de respuesta promedio entre las aplicaciones CRUD desarrolladas bajo los	1. Tipo de investigación La presente investigación es de tipo aplicada, debido a que busca resolver un problema concreto relacionado con el rendimiento en persistencia de datos de aplicaciones CRUD desarrolladas bajo dos paradigmas de programación: imperativo y reactivo, con el propósito de comparar las diferencias de rendimiento bajo las mismas condiciones de evaluación. 2. Nivel de investigación El nivel de investigación es descriptivo, ya que se orienta a medir, analizar y describir las características de rendimiento en persistencia de datos de aplicaciones CRUD desarrolladas bajo diferentes

operaciones de persistencia de datos?	programación imperativo y reactivo durante las operaciones de persistencia de datos.	paradigmas de programación imperativo y reactivo.	paradigmas de programación, considerando indicadores como tiempo de respuesta, throughput, latencia de operaciones y consumo de recursos.
b. ¿Qué diferencias existen en el consumo de recursos del sistema (CPU Y JVM) entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo?	b) Comparar el consumo de recursos del sistema (CPU Y JVM) de las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.	b) Existen diferencias en el consumo de recursos del sistema (CPU y memoria JVM) entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.	3. Enfoque de investigación El estudio se desarrolla bajo un enfoque cuantitativo, ya que los datos recogidos son numéricos, medibles y se analizarán mediante indicadores objetivos como tiempo de respuesta, consumo de recursos, latencia y throughput.
c. ¿Qué diferencias existen en la latencia de operación entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en los	c) Comparar la latencia en operación de las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo bajo distintos	c) Existen diferencias en la latencia promedio entre las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.	4. Diseño de investigación La presente investigación adopta un diseño descriptivo-comparativo, debido a que se desarrollan dos aplicaciones CRUD funcionalmente equivalentes bajo distintos paradigmas de programación imperativo y reactivo, y posteriormente se comparan sus métricas de rendimiento

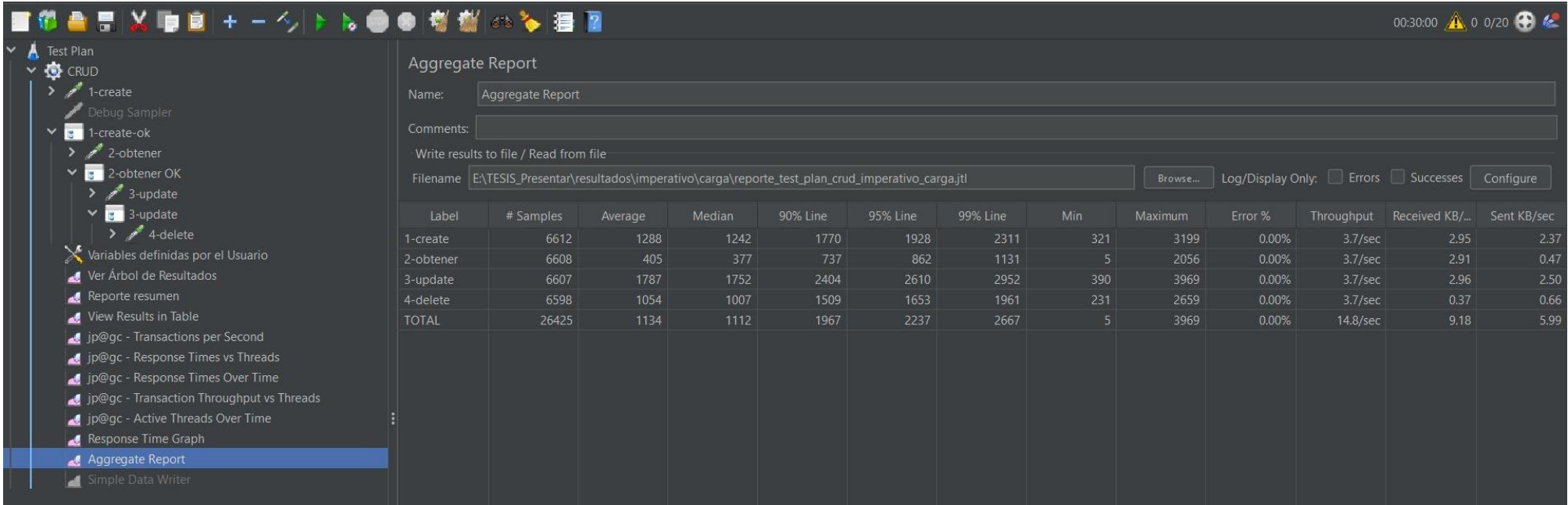
escenarios de pruebas de carga, estrés y picos?	escenarios de prueba de rendimiento.	d) Existen diferencias en el throughput alcanzado por las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo.	bajo las mismas condiciones de evaluación.
d. ¿Qué diferencias existen en el throughput alcanzado por las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo en los escenarios de pruebas de carga, estrés y picos?	d) Medir y comparar el throughput alcanzado por las aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo durante las operaciones CRUD.	VARIABLE E INDICADORES	5. Población y muestra
		Variable de estudio Rendimiento en la persistencia de datos. Dimensiones 1. Tiempo de respuesta: Es el intervalo de tiempo que transcurre desde que se envía una solicitud de servicio hasta que la aplicación devuelve la respuesta. 2. Consumo de recursos: Es considerado en que las cantidades y tipos de recursos (CPU, memoria,	Por la naturaleza del diseño descriptivo-comparativo, la presente investigación no trabajó con una población ni con una muestra en sentido probabilístico, dado que no se pretende generalizar los resultados hacia un conjunto mayor de aplicaciones o ejecuciones. El análisis se efectuó de manera censal, sobre la totalidad de las mediciones de rendimiento registradas durante la evaluación de las dos aplicaciones CRUD desarrolladas bajo los paradigmas de programación imperativo y reactivo. En total se registraron 158 909 mediciones, distribuidas en 72 702 correspondientes a la aplicación

		ancho de banda de red, etc.) que emplea un producto. 3. Throughput: Se define como la cantidad de transacción que son transferidos exitosamente de un remitente a un receptor a través de una red o un enlace de comunicación. 4. Latencia de operaciones: Se define el tiempo de operaciones de lectura y escritura, de una consulta a la base de datos en procesar y devolver el resultado hasta recibir el 1er byte.	imperativa y 86 207 a la aplicación reactiva, obtenidas en los escenarios de pruebas de carga, estrés y picos ejecutados con Apache JMeter desde 1 hasta 20 usuarios virtuales durante 30 minutos por escenario, conforme al siguiente detalle: <table><tr><th>Escenario</th><th>Aplicación imperativa</th><th>Aplicación reactiva</th></tr><tr><td>Prueba de carga</td><td>26 425</td><td>31 497</td></tr><tr><td>Prueba de estrés</td><td>19 878</td><td>24 672</td></tr><tr><td>Prueba de picos</td><td>26 399</td><td>30 038</td></tr><tr><td>Total</td><td>72 702</td><td>86 207</td></tr></table> Cada medición corresponde a la ejecución de una operación CRUD (Create, Read, Update, Delete), y sobre	Escenario	Aplicación imperativa	Aplicación reactiva	Prueba de carga	26 425	31 497	Prueba de estrés	19 878	24 672	Prueba de picos	26 399	30 038	Total	72 702	86 207
Escenario	Aplicación imperativa	Aplicación reactiva																
Prueba de carga	26 425	31 497																
Prueba de estrés	19 878	24 672																
Prueba de picos	26 399	30 038																
Total	72 702	86 207																

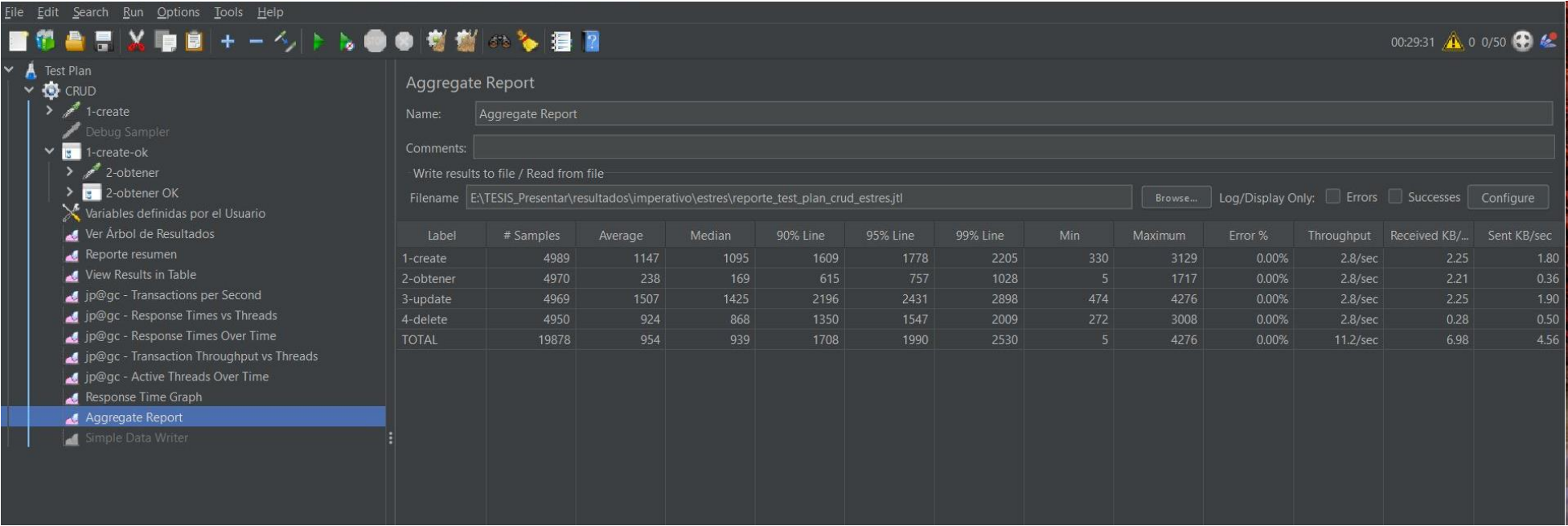
			este conjunto completo de observaciones se calcularon las métricas de tiempo de respuesta, latencia, throughput y consumo de recursos (CPU y memoria JVM), por operación, escenario de prueba y aplicación evaluada 6. Técnicas de recolección de datos Pruebas de rendimiento: Son técnicas que miden la estabilidad, velocidad y capacidad de un sistema para mantener un nivel óptimo de funcionamiento bajo condiciones controlada. A través de estas pruebas se recopiló información sobre tiempos de respuesta, consumo de recursos, throughput y latencia, permitiendo comparar el comportamiento de ambas aplicaciones bajo las mismas condiciones. En este estudio se aplicaron tres modalidades: - Prueba de carga - Prueba de estrés - Prueba de picos 7. Instrumentos de recolección de datos
--	--	--	--

			Se utilizó Apache JMeter como instrumento principal para la generación de carga y la recolección de métricas de rendimiento. Adicionalmente, se empleó Grafana para la visualización y monitoreo de métricas durante la ejecución de las pruebas.
--	--	--	---

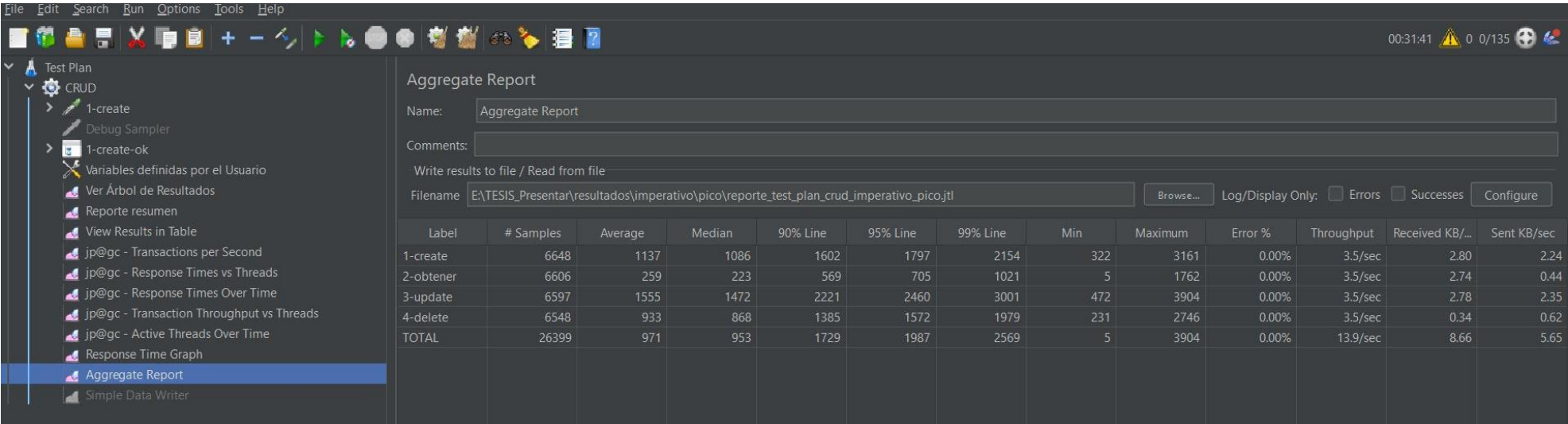
Anexo B: Resultados de los resultados de la prueba de carga del aplicativo sincrónico



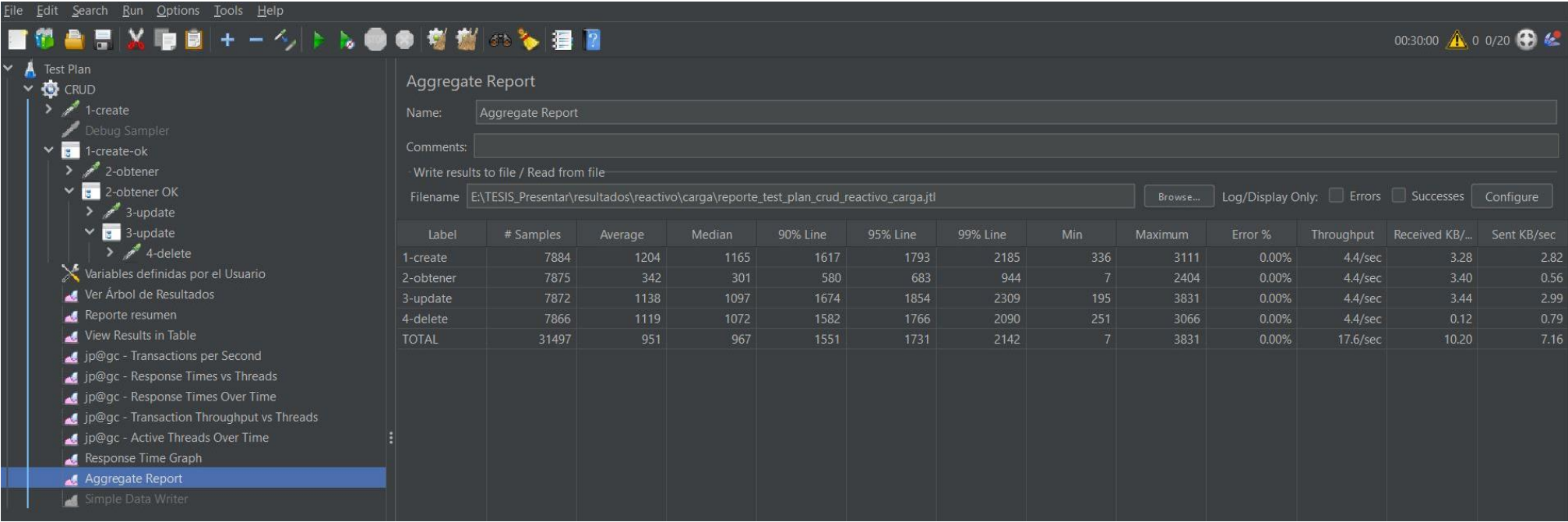
Anexo C: Resultados de los resultados de la prueba de estrés del aplicativo sincrónico



Anexo D: Resultados de los resultados de la prueba de picos del aplicativo sincrónico



ANEXO B: Resultados de los resultados de la prueba de carga del aplicativo asincrónico



Anexo C: Resultados de los resultados de la prueba de estrés del aplicativo asincrónico

File Edit Search Run Options Tools Help

00:29:30 0 0/50

Test Plan

- CRUD
 - 1-create
 - Debug Sampler
 - 1-create-ok
 - 2-obtener
 - 2-obtener OK
 - 3-update
 - 3-update
 - 4-delete
 - Variables definidas por el Usuario
 - Ver Árbol de Resultados
 - Reporte resumen
 - View Results in Table
 - jp@gc - Transactions per Second
 - jp@gc - Response Times vs Threads
 - jp@gc - Response Times Over Time
 - jp@gc - Transaction Throughput vs Threads
 - jp@gc - Active Threads Over Time
 - Response Time Graph
 - Aggregate Report**
 - Simple Data Writer

Aggregate Report

Name: Aggregate Report

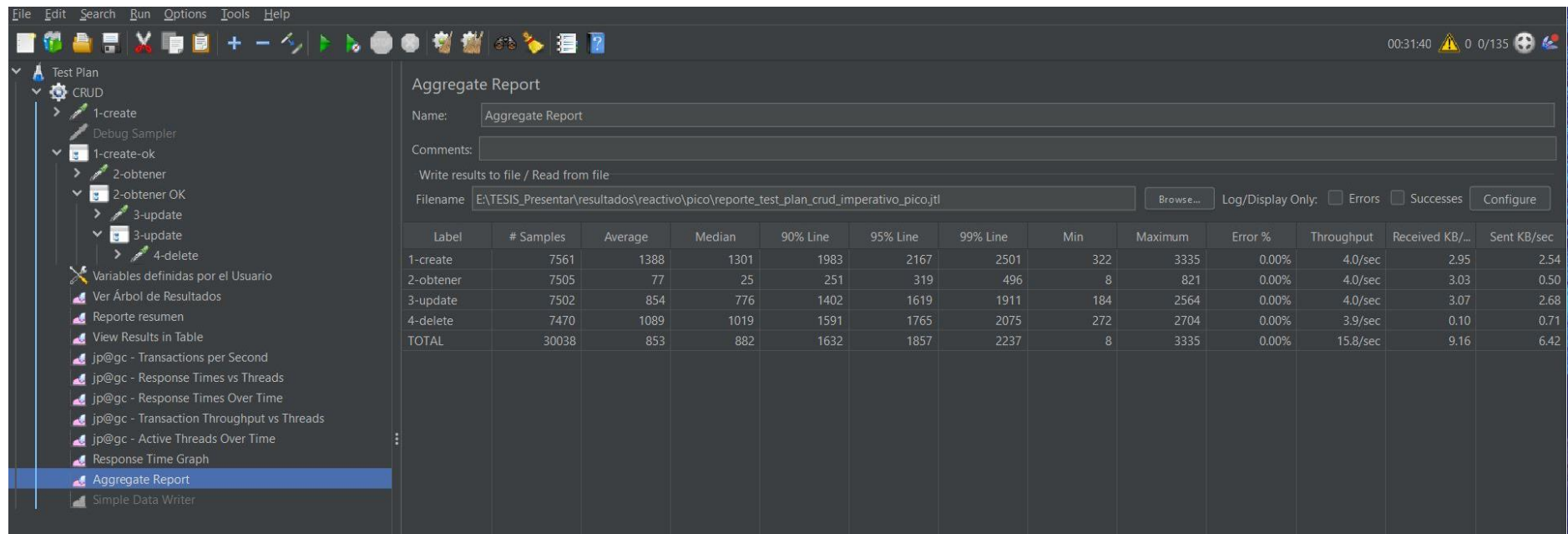
Comments:

Write results to file / Read from file

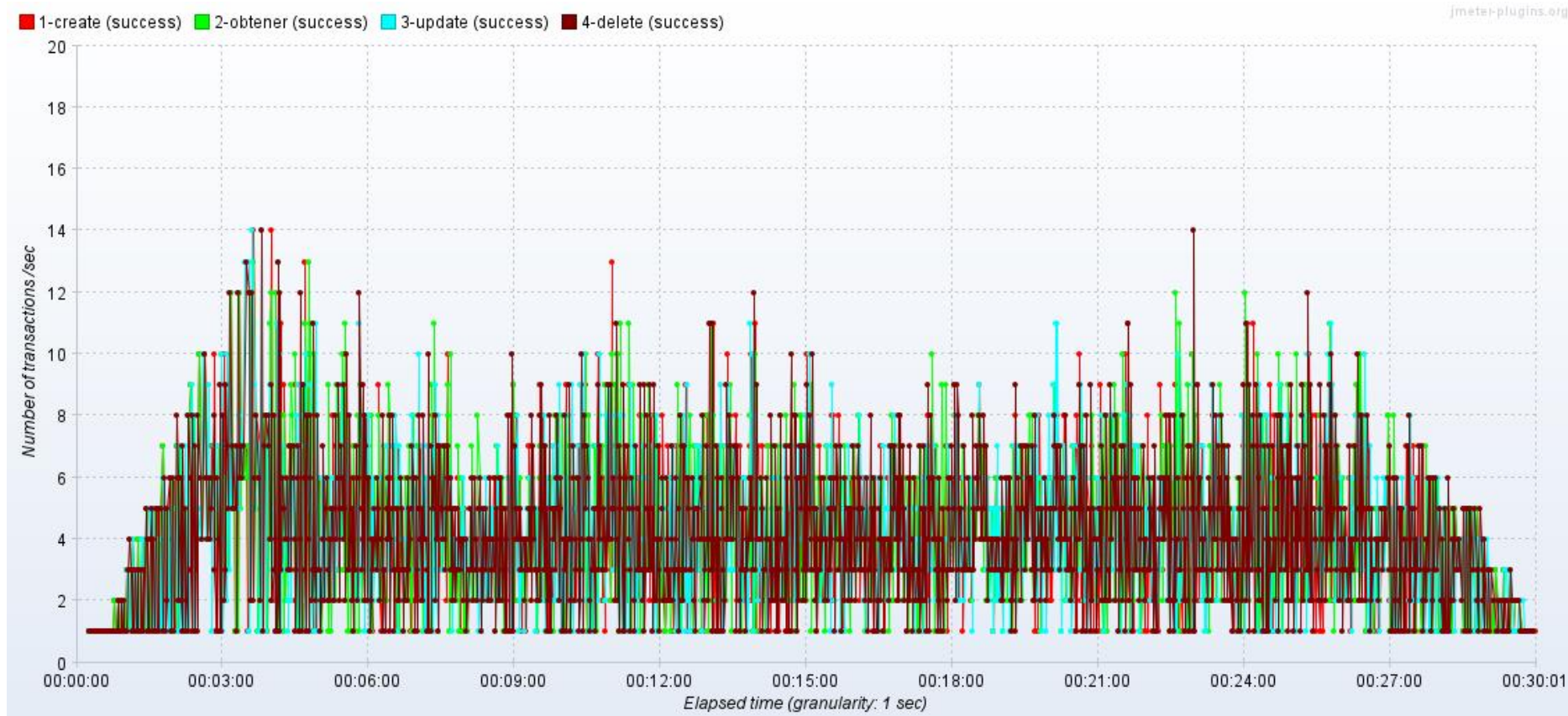
Filename: E:\TESIS_Presentar\resultados\reactivo\estres\reporte_test_plan_crud_estres.jtl Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
1-create	6189	1077	1041	1475	1606	1914	310	5544	0.00%	3.5/sec	2.60	2.24
2-obtener	6167	265	257	452	557	812	8	2296	0.00%	3.5/sec	2.68	0.44
3-update	6164	842	760	1381	1559	1927	167	4327	0.00%	3.5/sec	2.71	2.36
4-delete	6152	892	834	1316	1484	1797	240	4321	0.00%	3.5/sec	0.09	0.63
TOTAL	24672	769	765	1322	1499	1832	8	5544	0.00%	14.0/sec	8.07	5.66

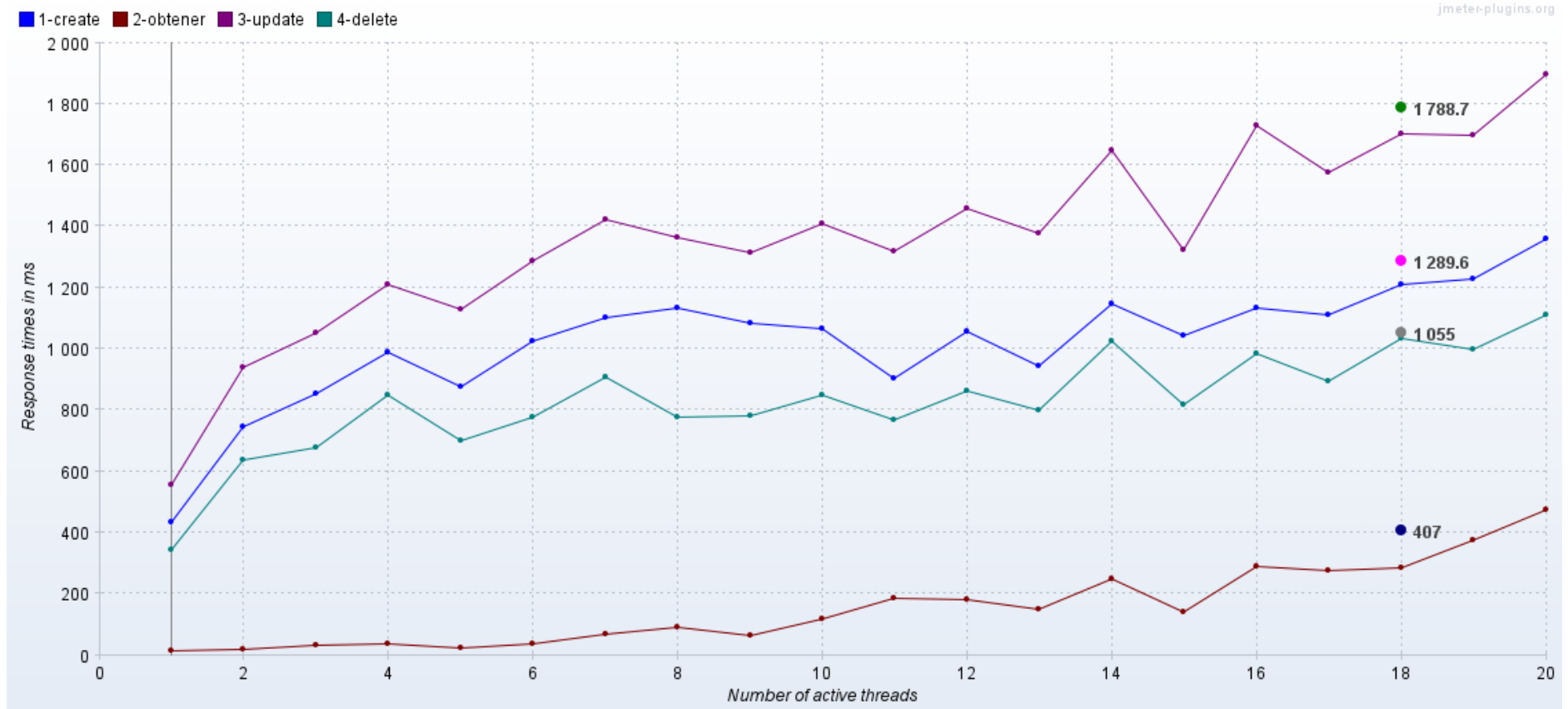
Anexo D: Resultados de los resultados de la prueba de picos del aplicativo asincrónico



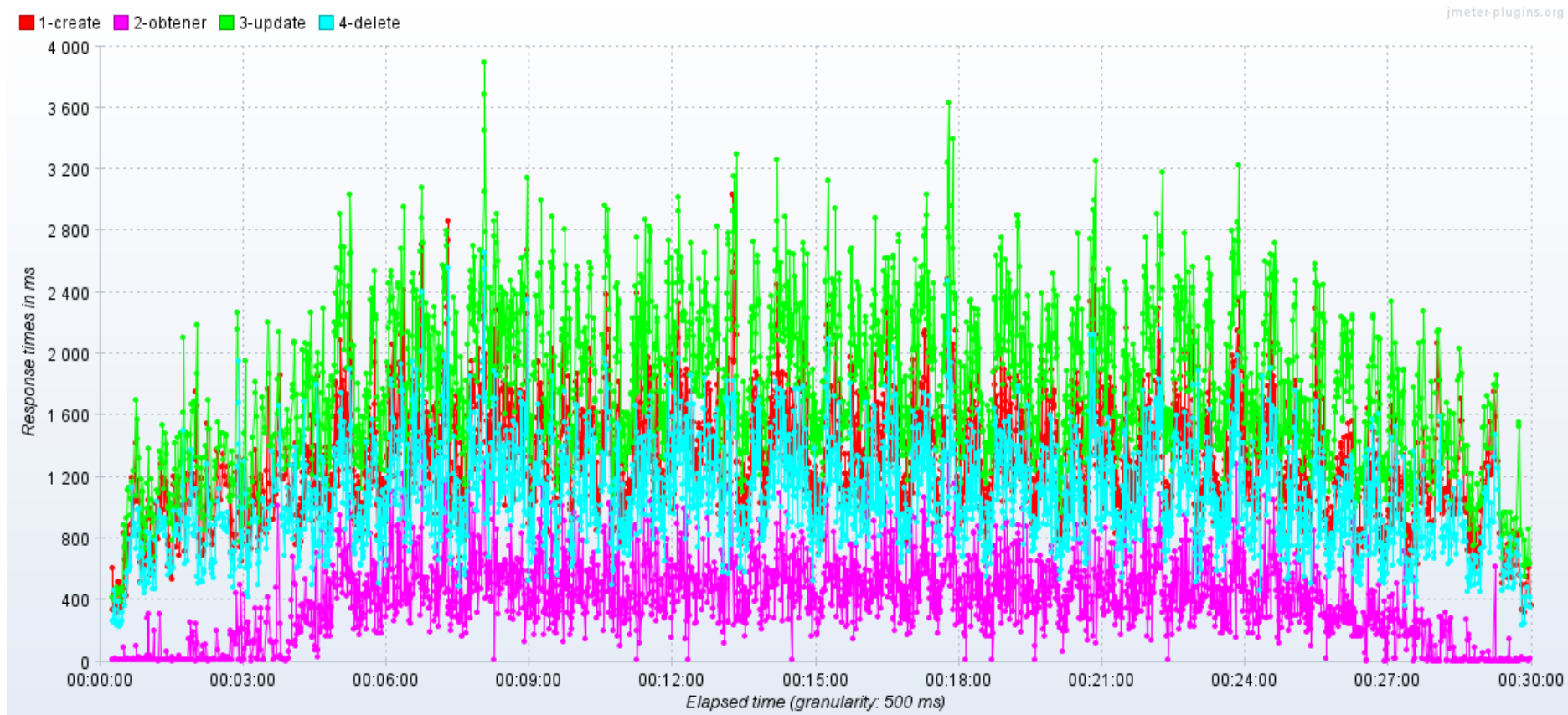
Anexo E: Resultados de throughput en la prueba de carga del aplicativo sincrónico



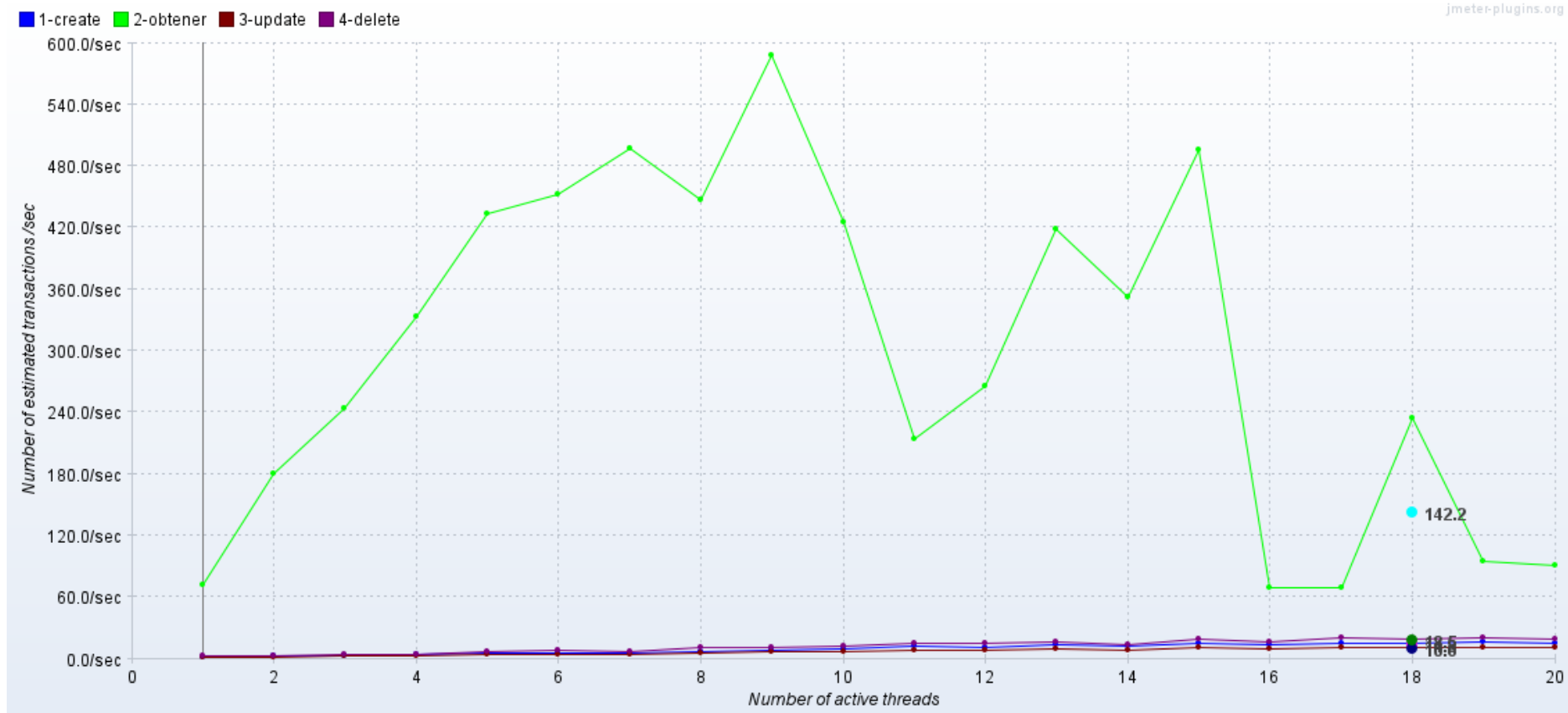
Anexo F: Resultados de tiempo de respuesta vs usuarios en la prueba de carga del aplicativo sincrónico



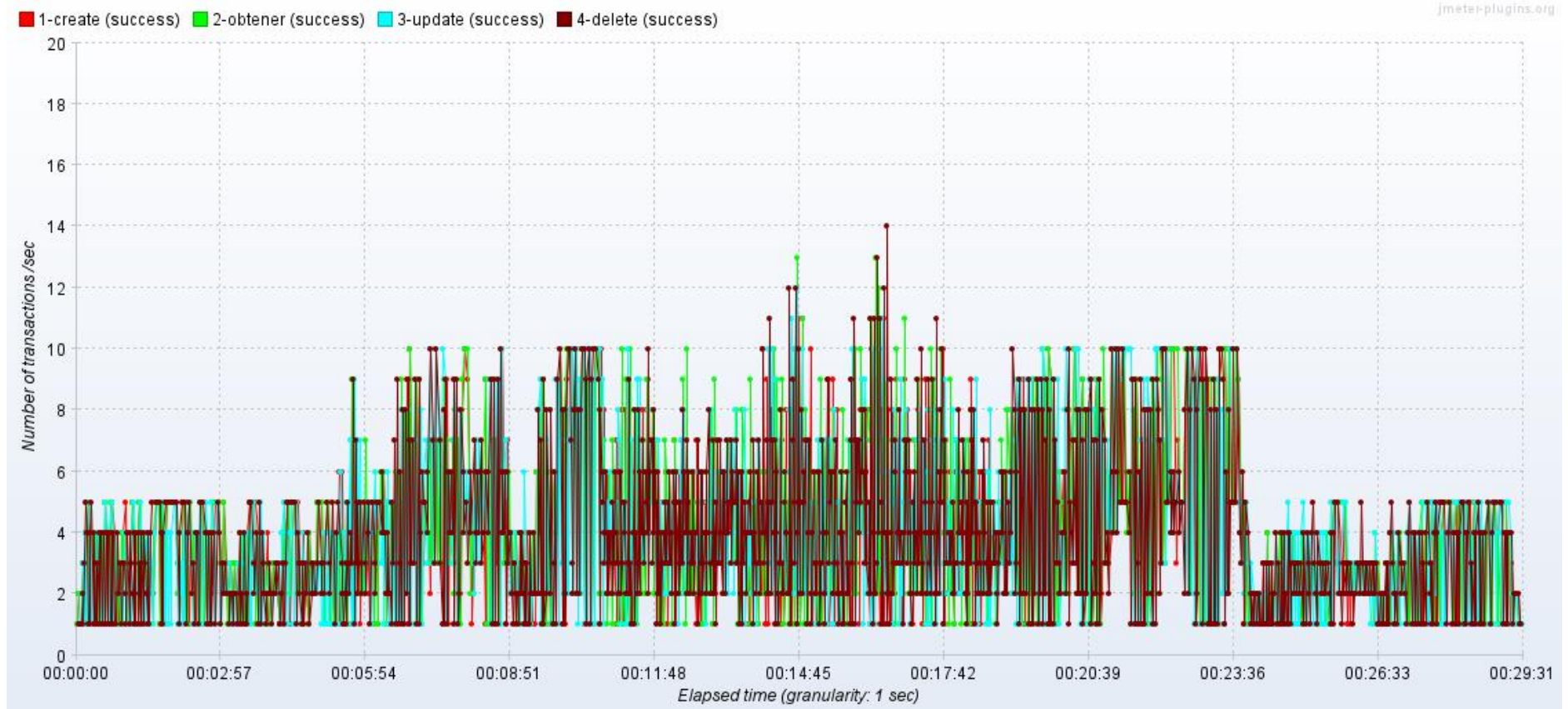
Anexo G: Resultados de tiempo de respuesta en la prueba de carga del aplicativo sincrónico



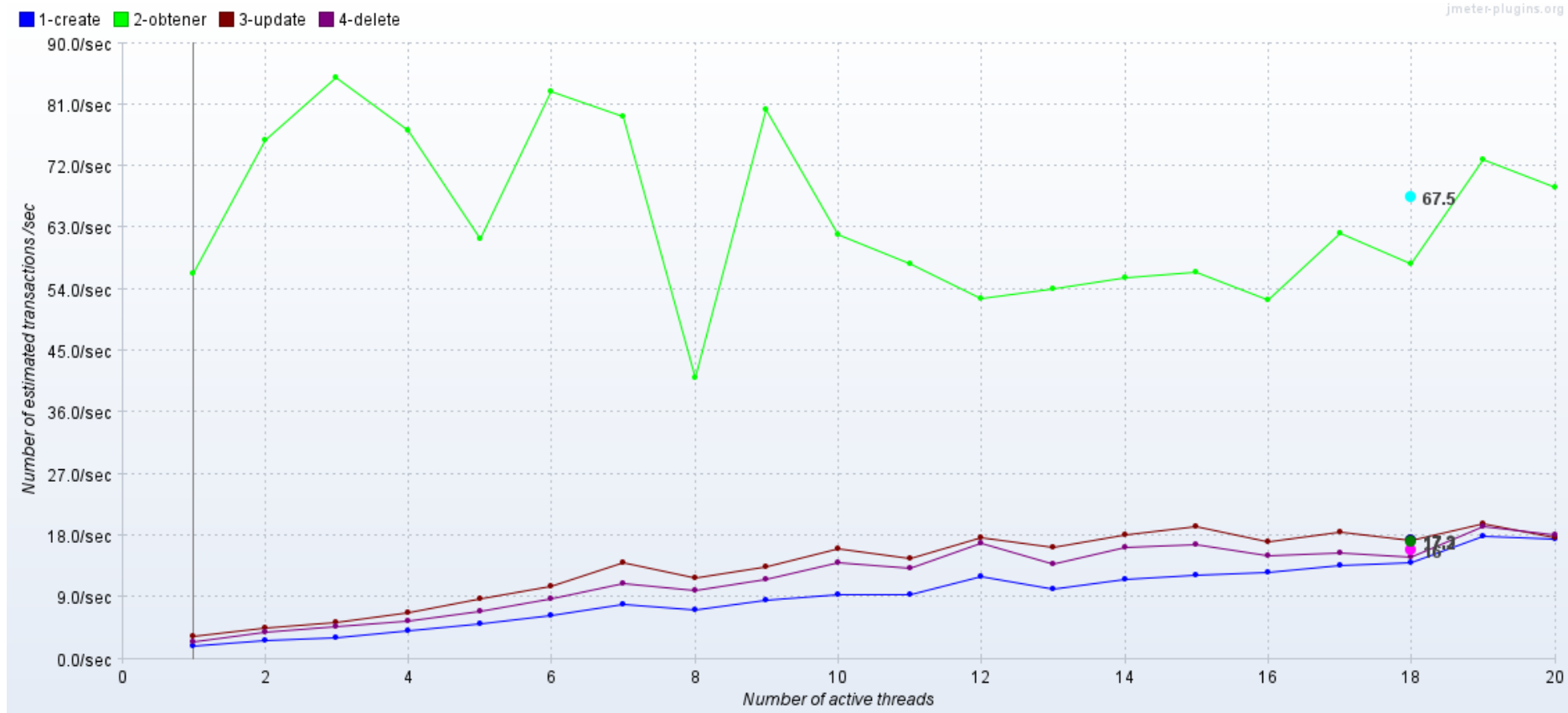
Anexo H: Resultados de TPS vs usuarios en la prueba de carga del aplicativo sincrónico



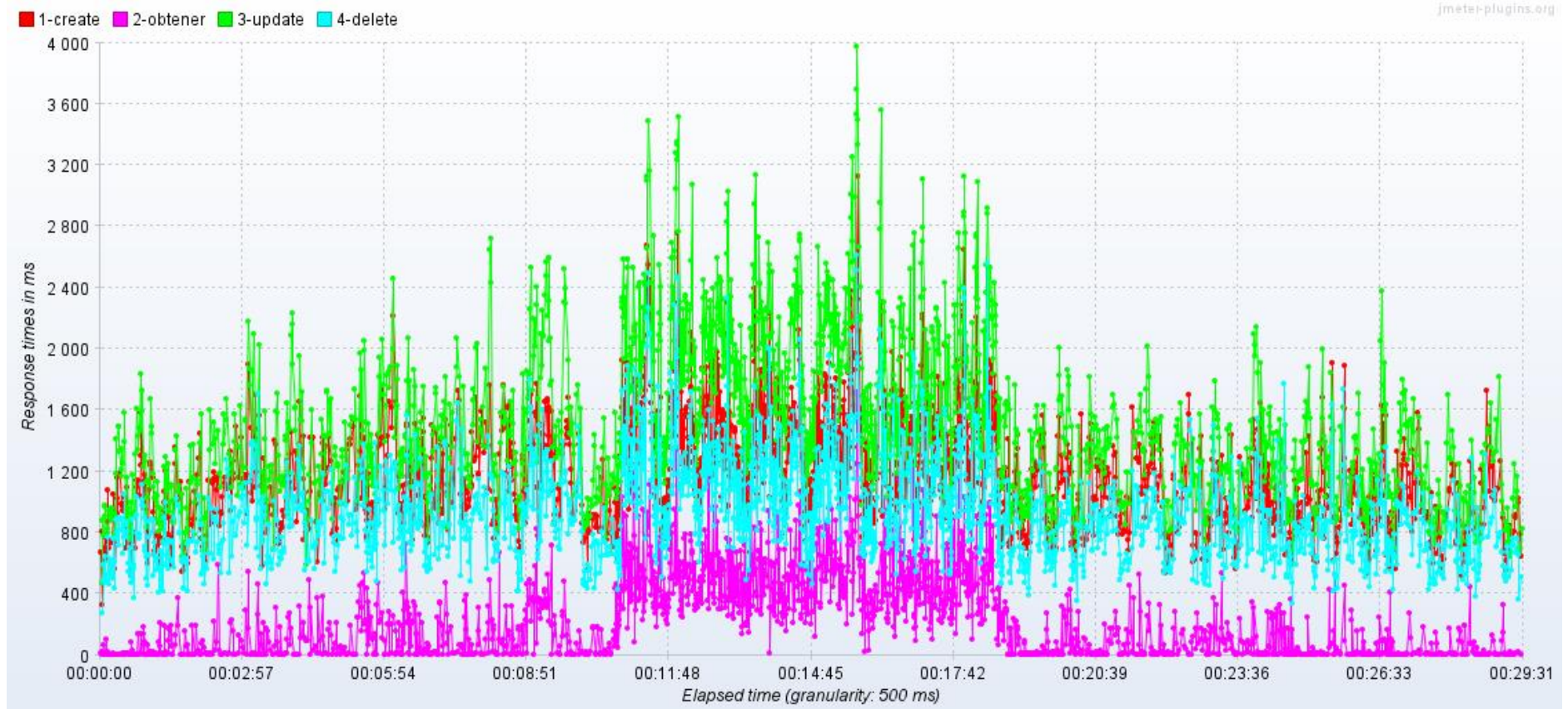
Anexo I: Resultados de throughput en la prueba de estrés del aplicativo sincrónico



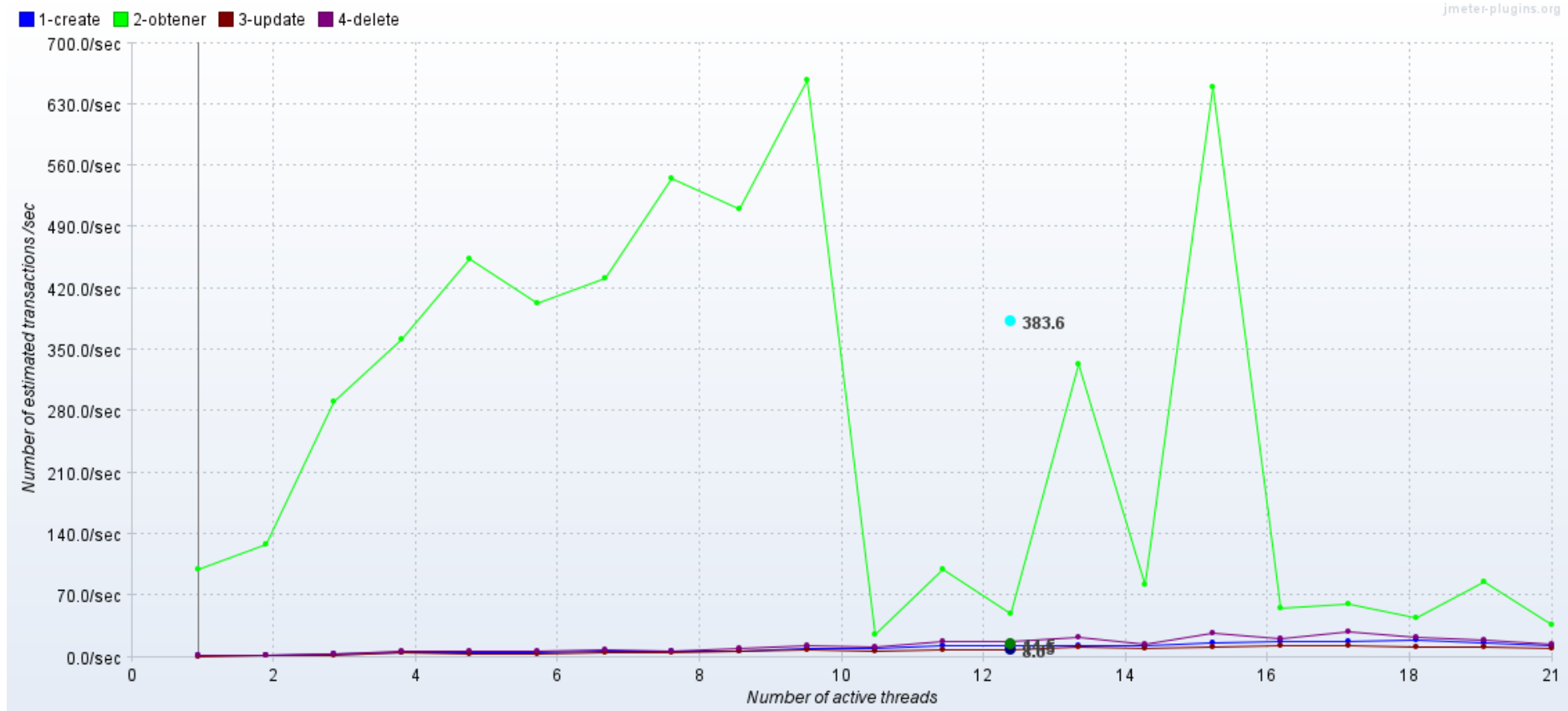
Anexo J: Resultados de tiempo de respuesta vs usuarios en la prueba de estrés del aplicativo sincrónico



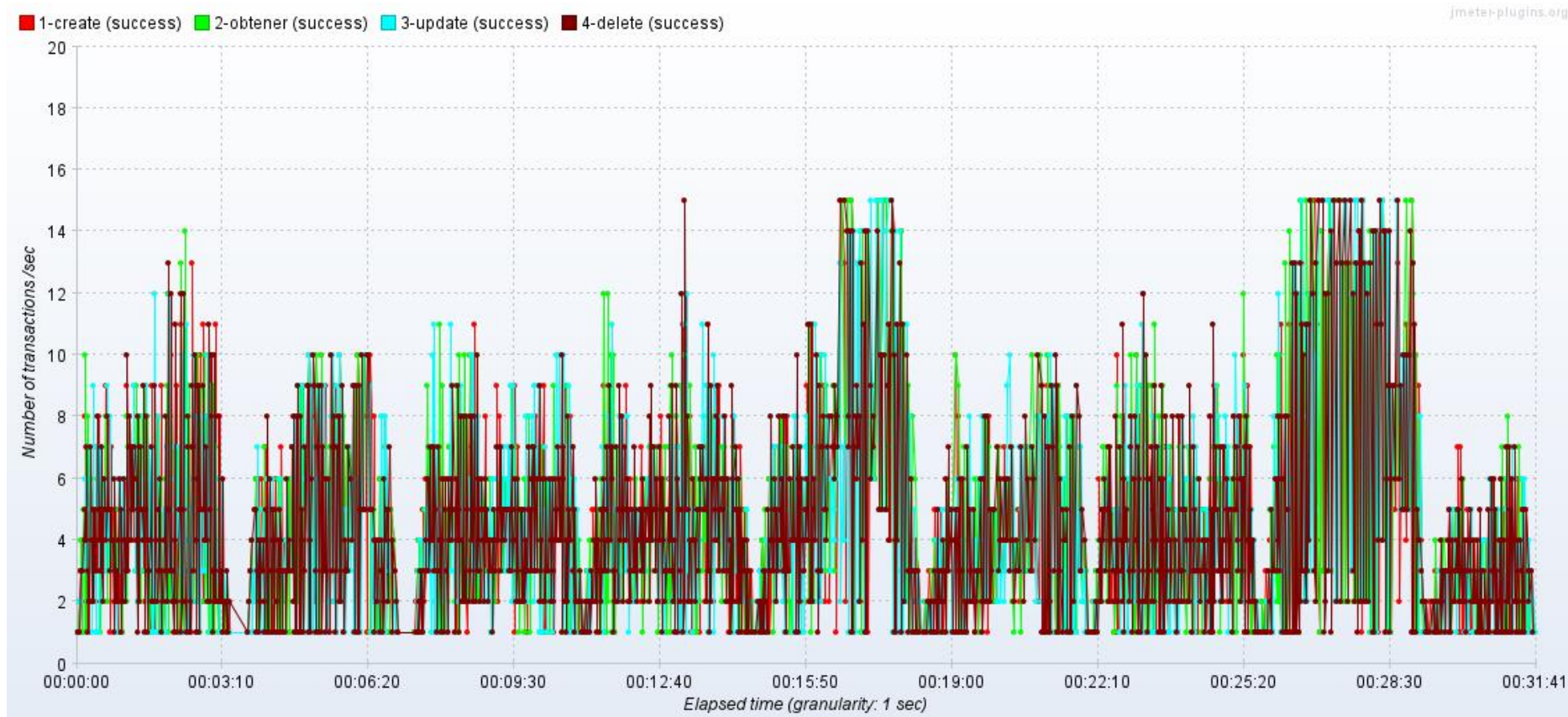
Anexo K: Resultados de tiempo de respuesta en la prueba de estrés del aplicativo sincrónico



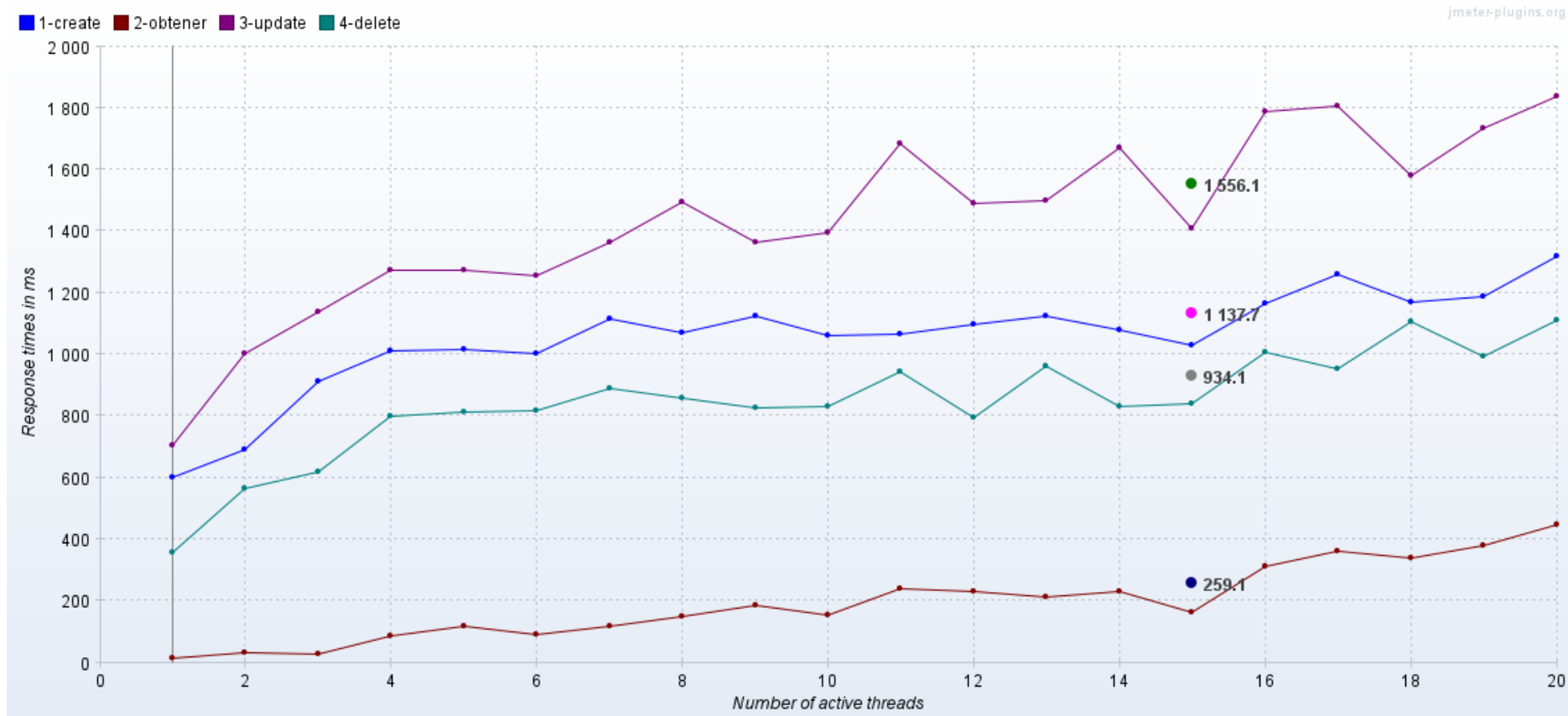
Anexo L: Resultados de TPS vs usuarios en la prueba de estrés del aplicativo sincrónico



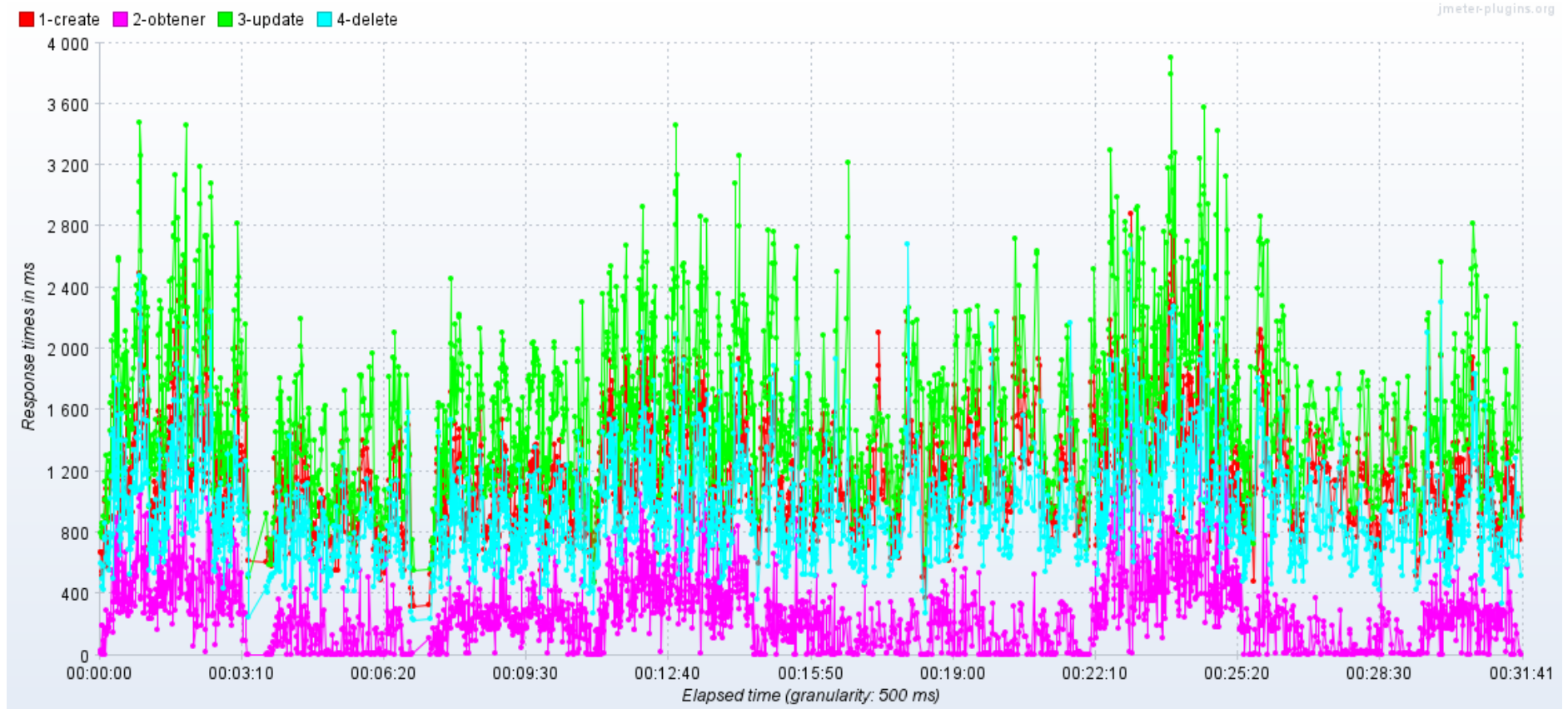
Anexo LL: Resultados de throughput en la prueba de picos del aplicativo sincrónico



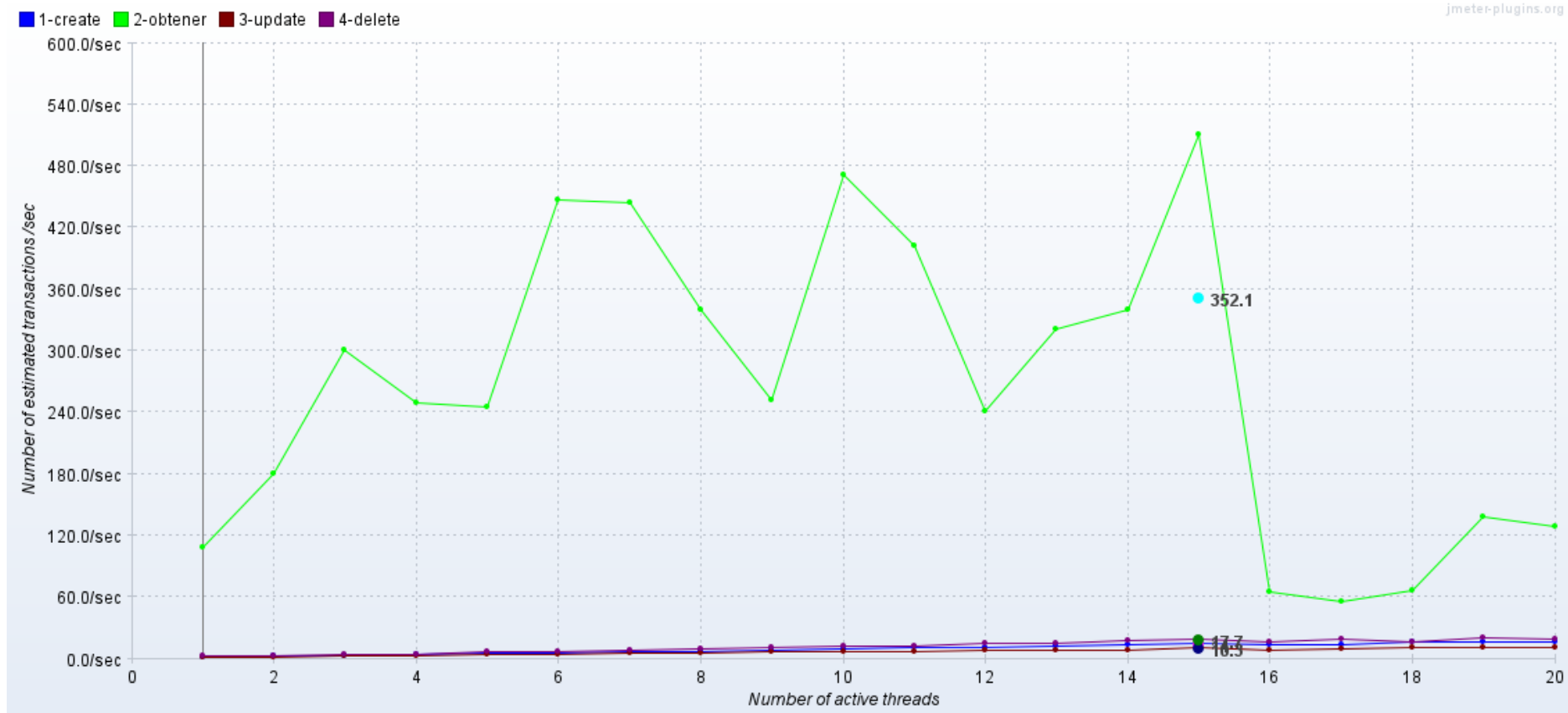
Anexo M: Resultados de tiempo de respuesta vs usuarios en la prueba de picos del aplicativo sincrónico



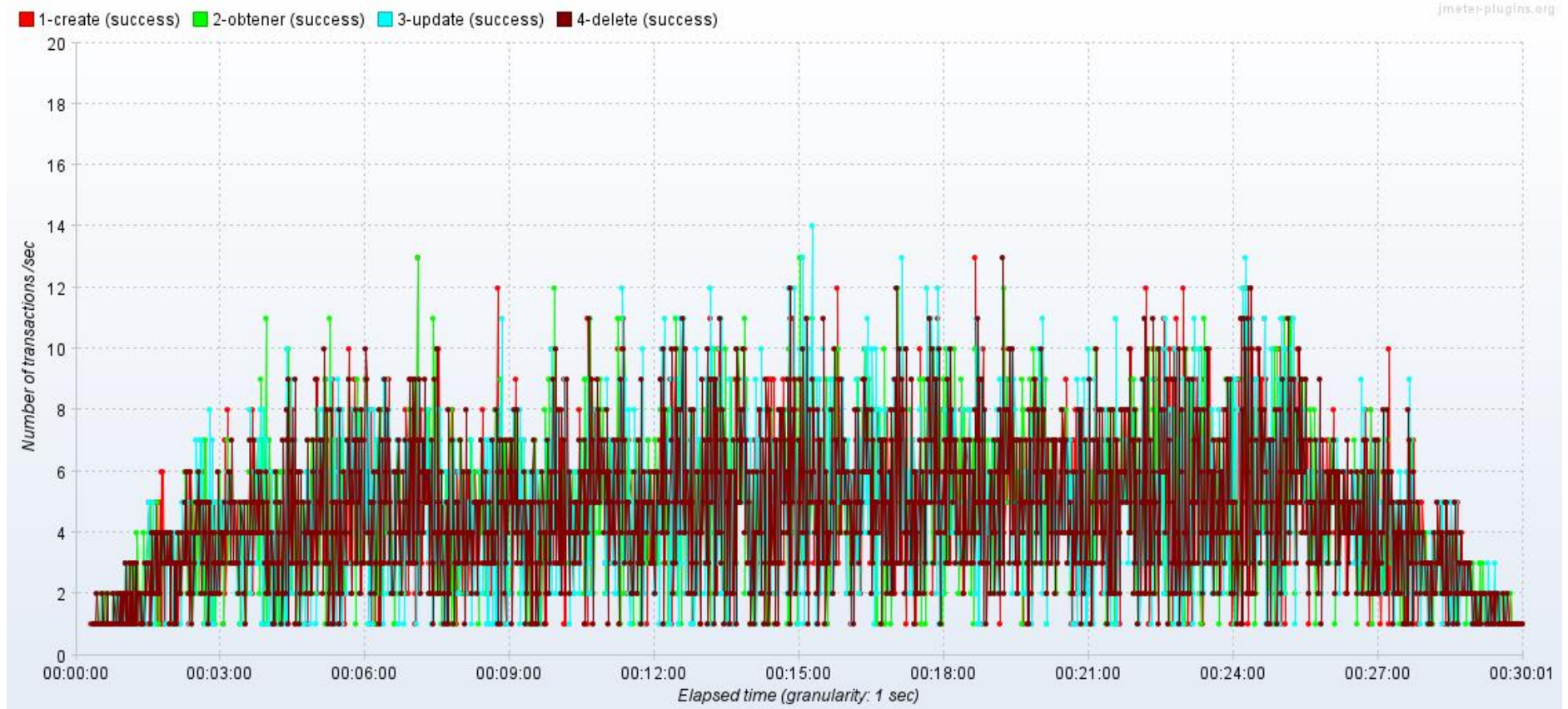
Anexo N: Resultados de tiempo de respuesta en la prueba de picos del aplicativo sincrónico



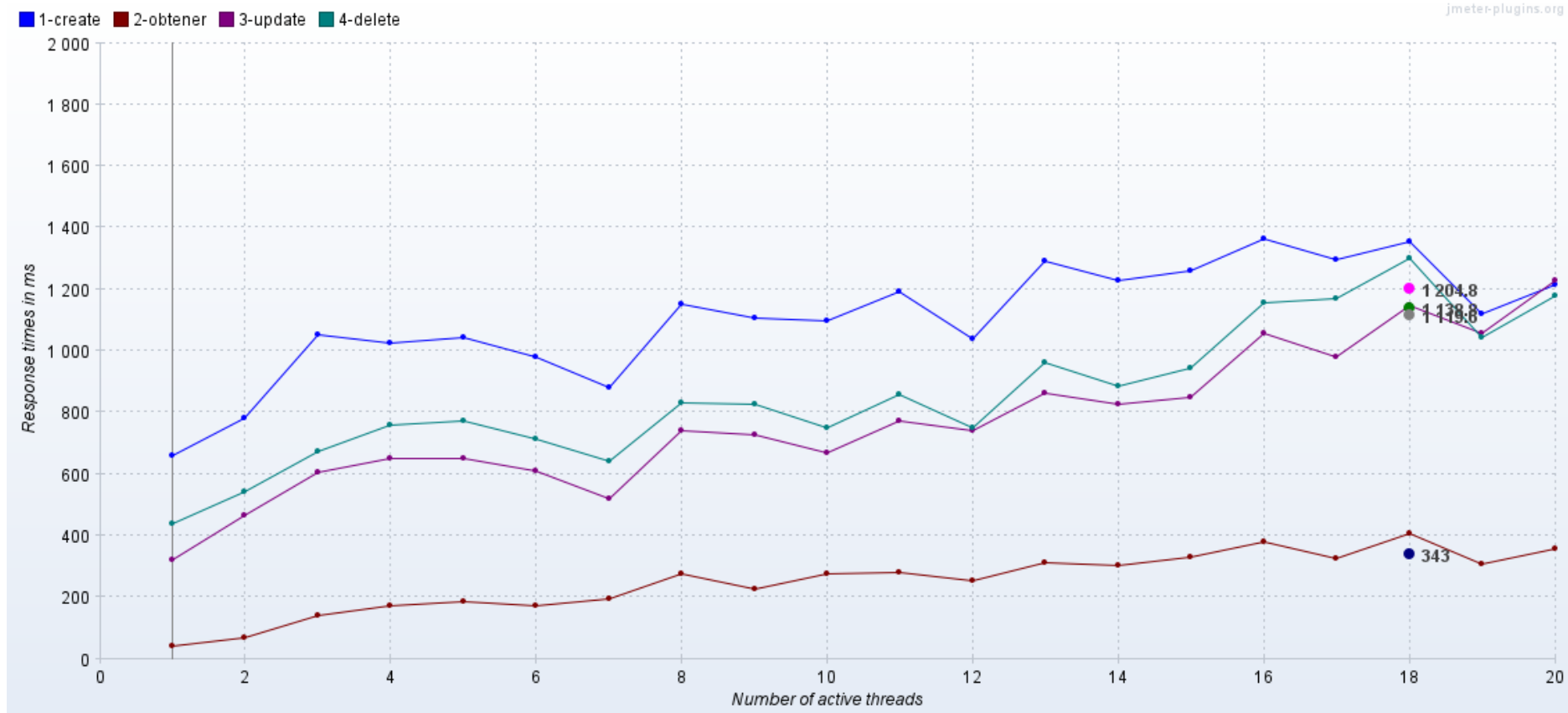
Anexo Ñ: Resultados de TPS vs usuarios en la prueba de picos del aplicativo sincrónico



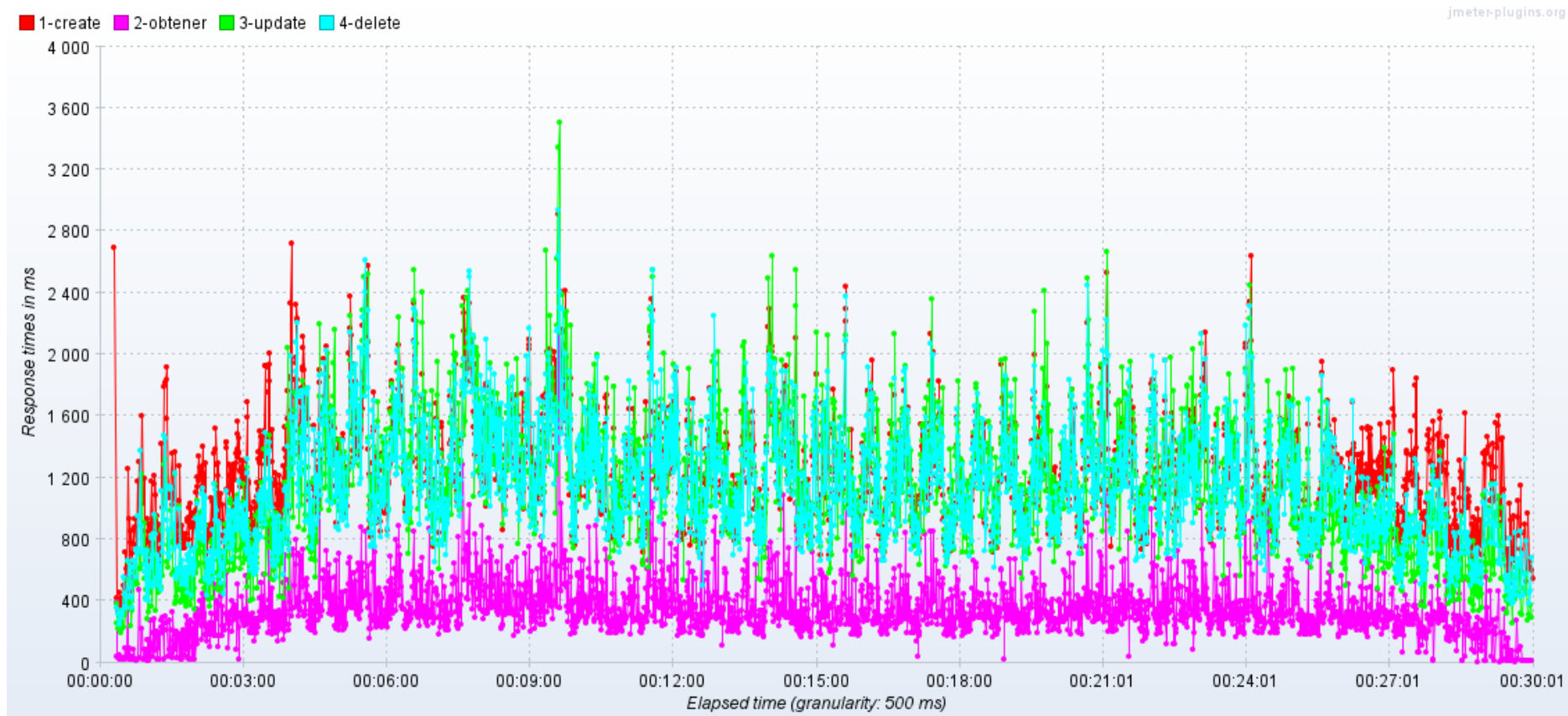
Anexo O: Resultados de throughput en la prueba de carga del aplicativo asincrónico



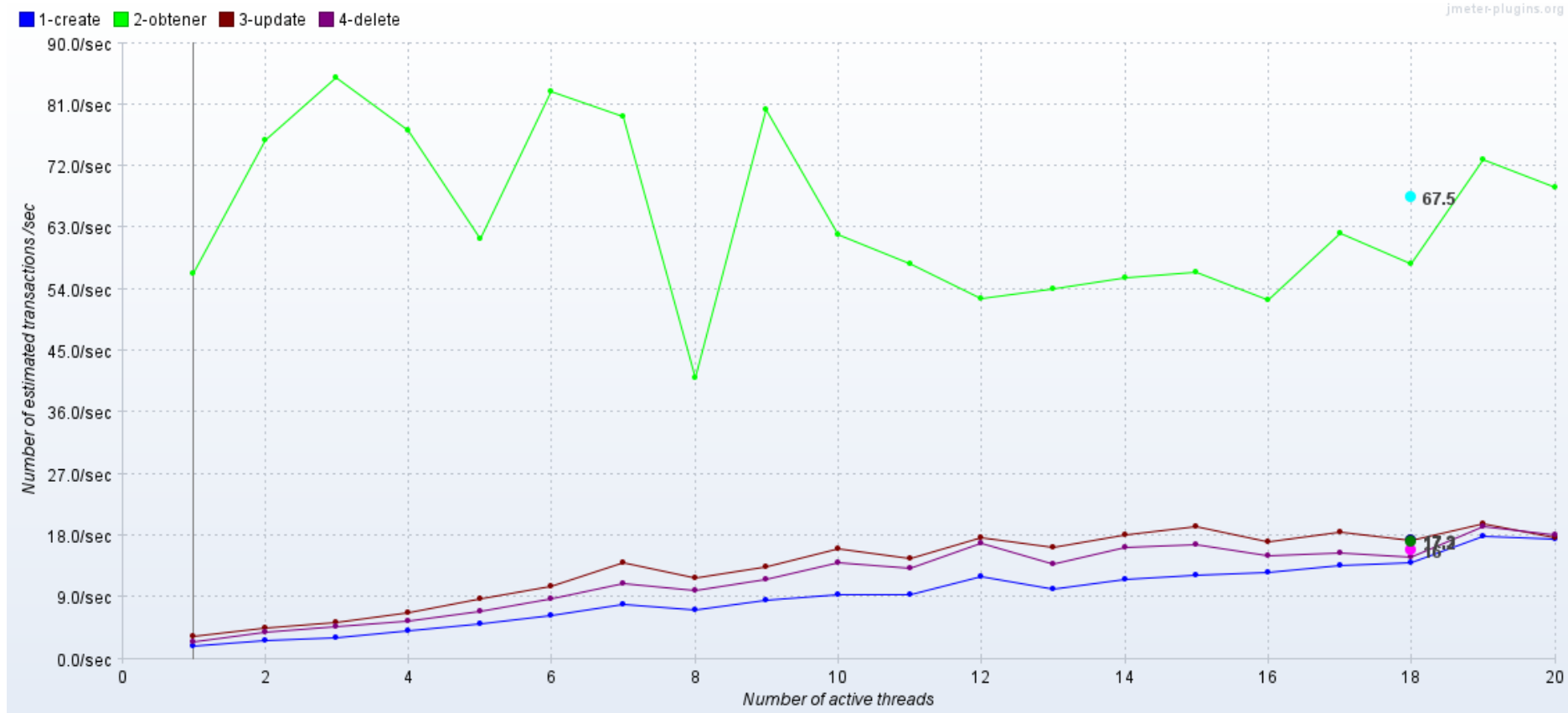
Anexo P: Resultados de tiempo de respuesta vs usuarios en la prueba de carga del aplicativo asincrónico



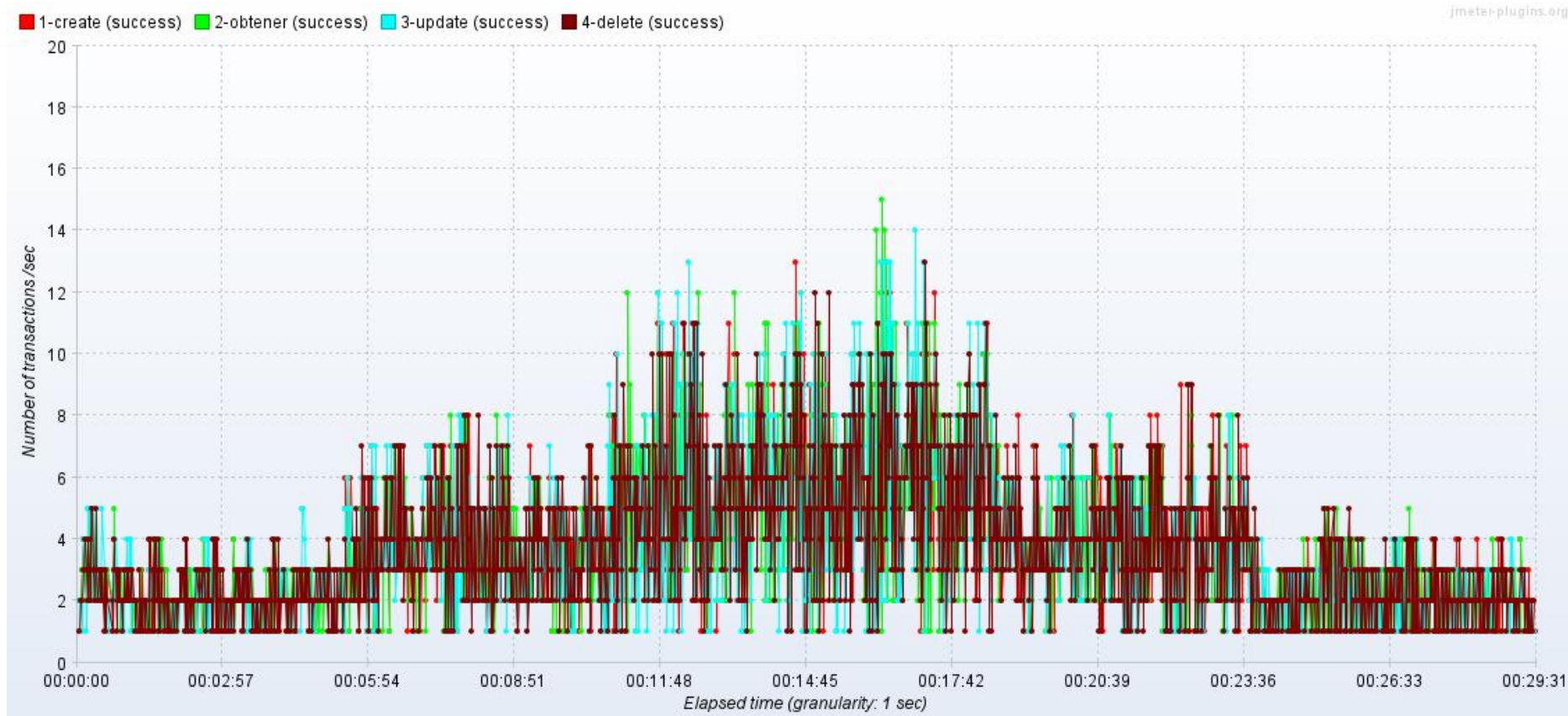
Anexo Q: Resultados de tiempo de respuesta en la prueba de carga del aplicativo asincrónico



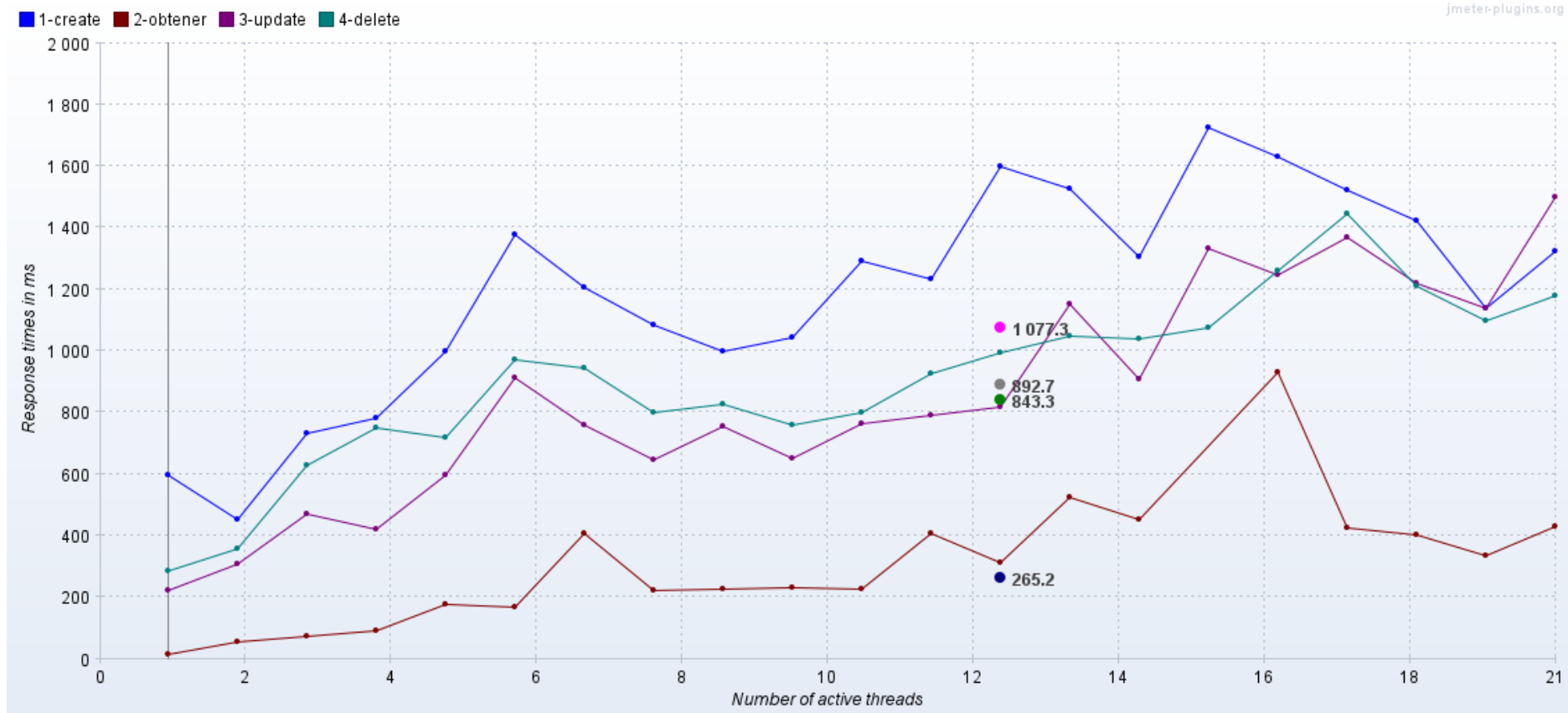
Anexo R: Resultados de TPS vs usuarios en la prueba de carga del aplicativo asincrónico



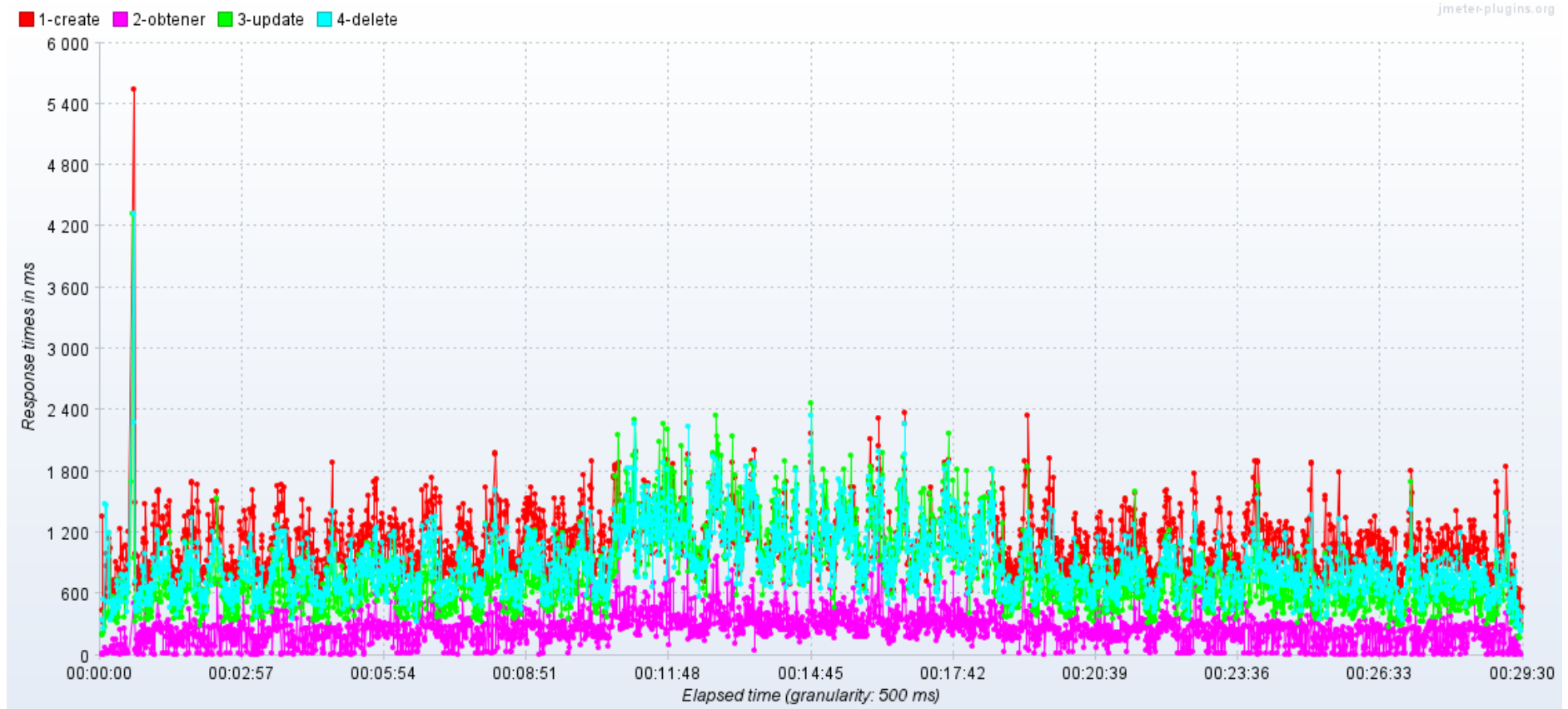
Anexo S: Resultados de throughput en la prueba de estrés del aplicativo asincrónico



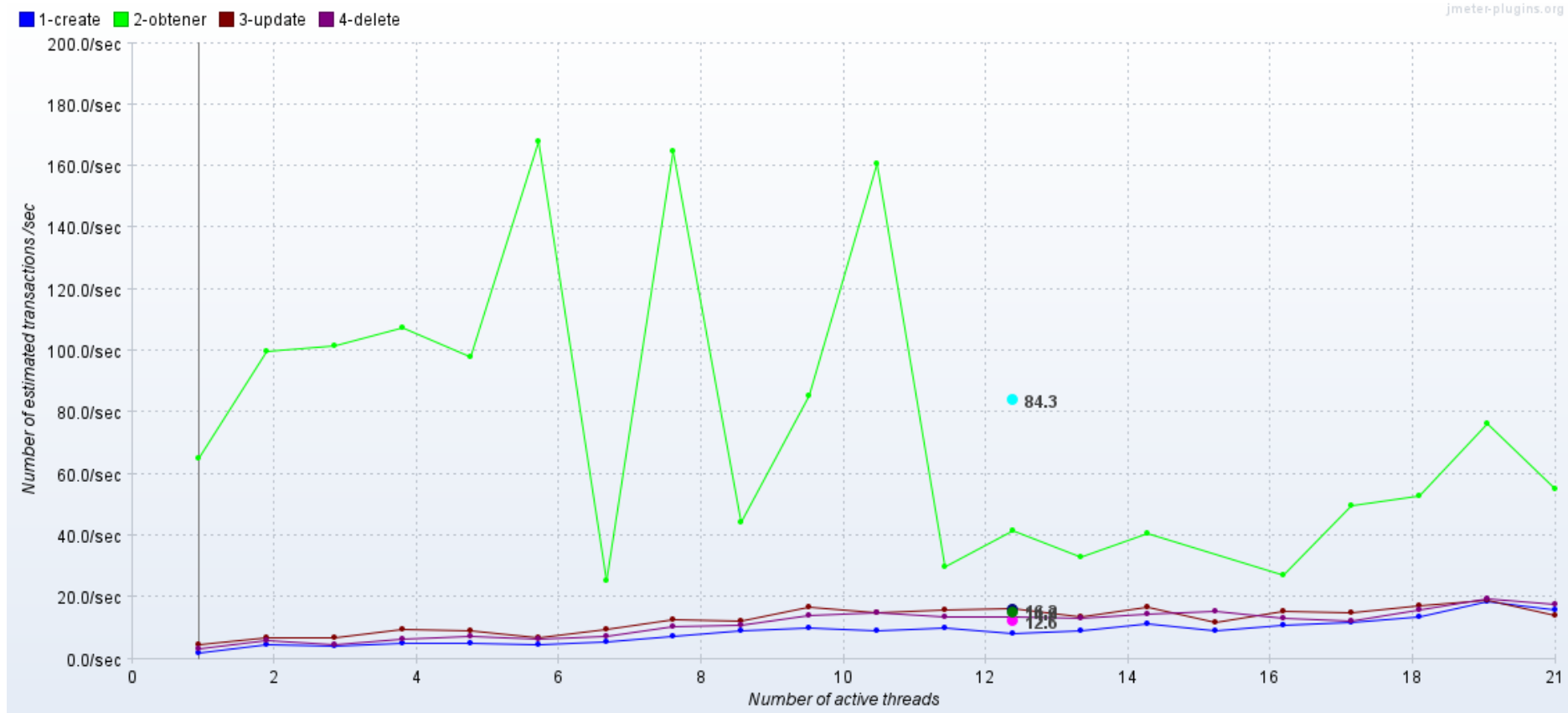
Anexo T: Resultados de tiempo de respuesta vs usuarios en la prueba de estrés del aplicativo asincrónico



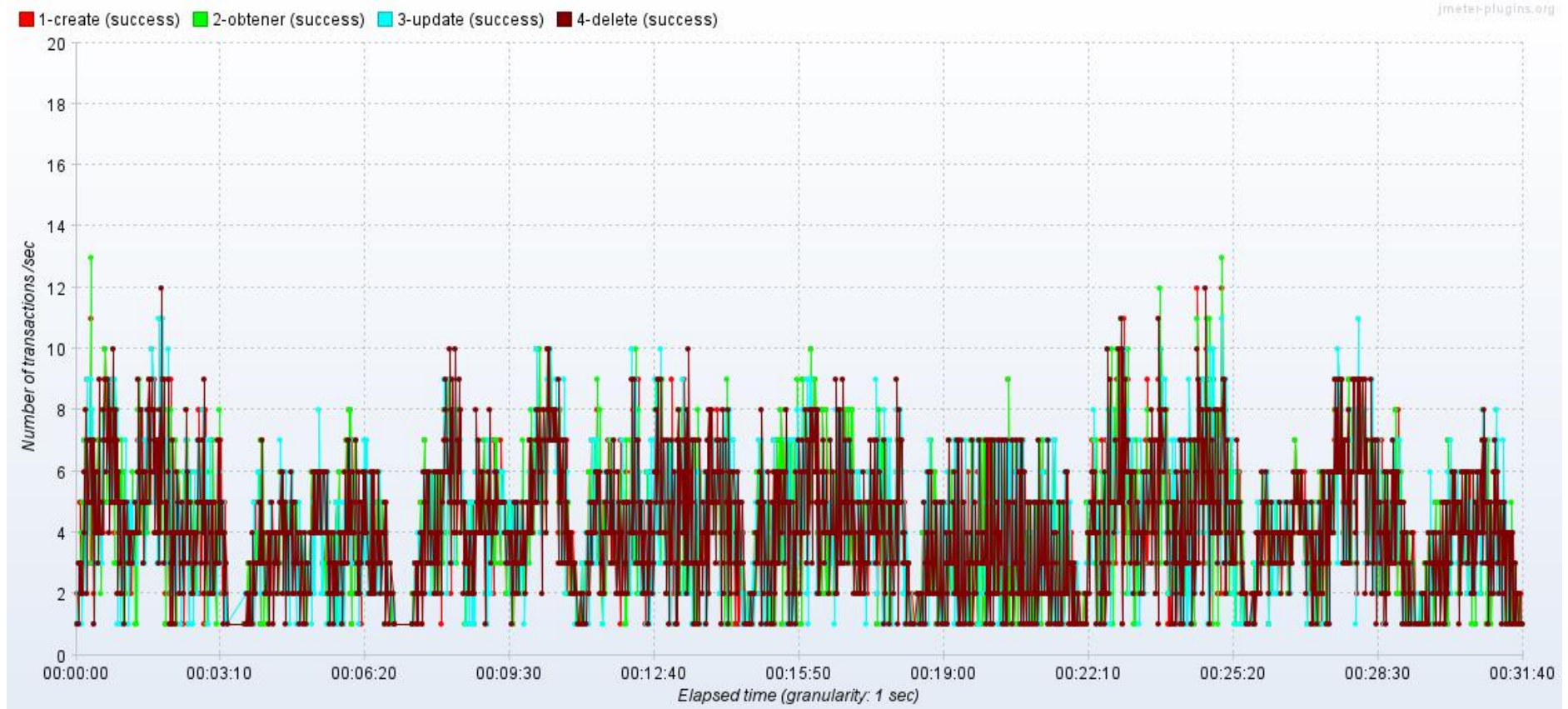
Anexo U: Resultados de tiempo de respuesta en la prueba de estrés del aplicativo asincrónico



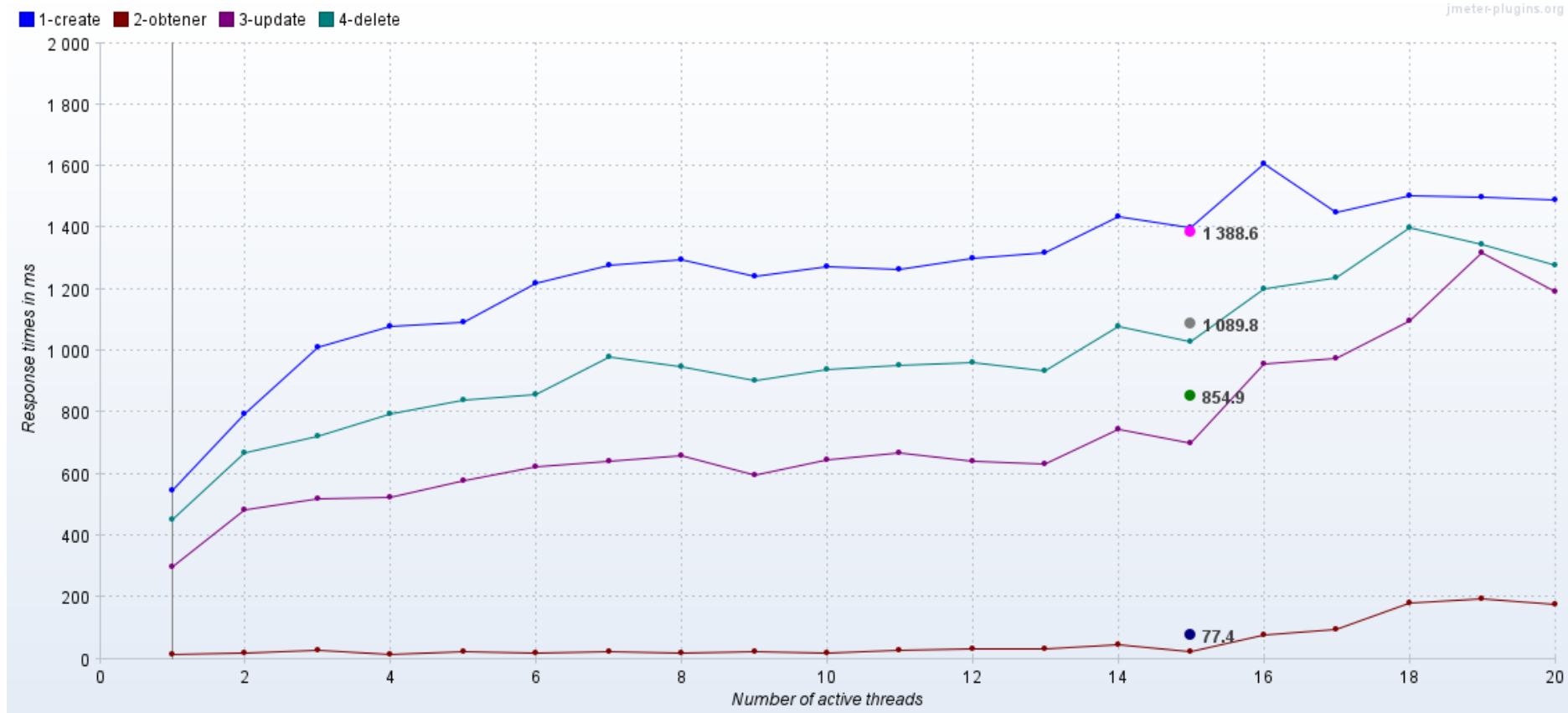
Anexo V: Resultados de TPS vs usuarios en la prueba de estrés del aplicativo asincrónico



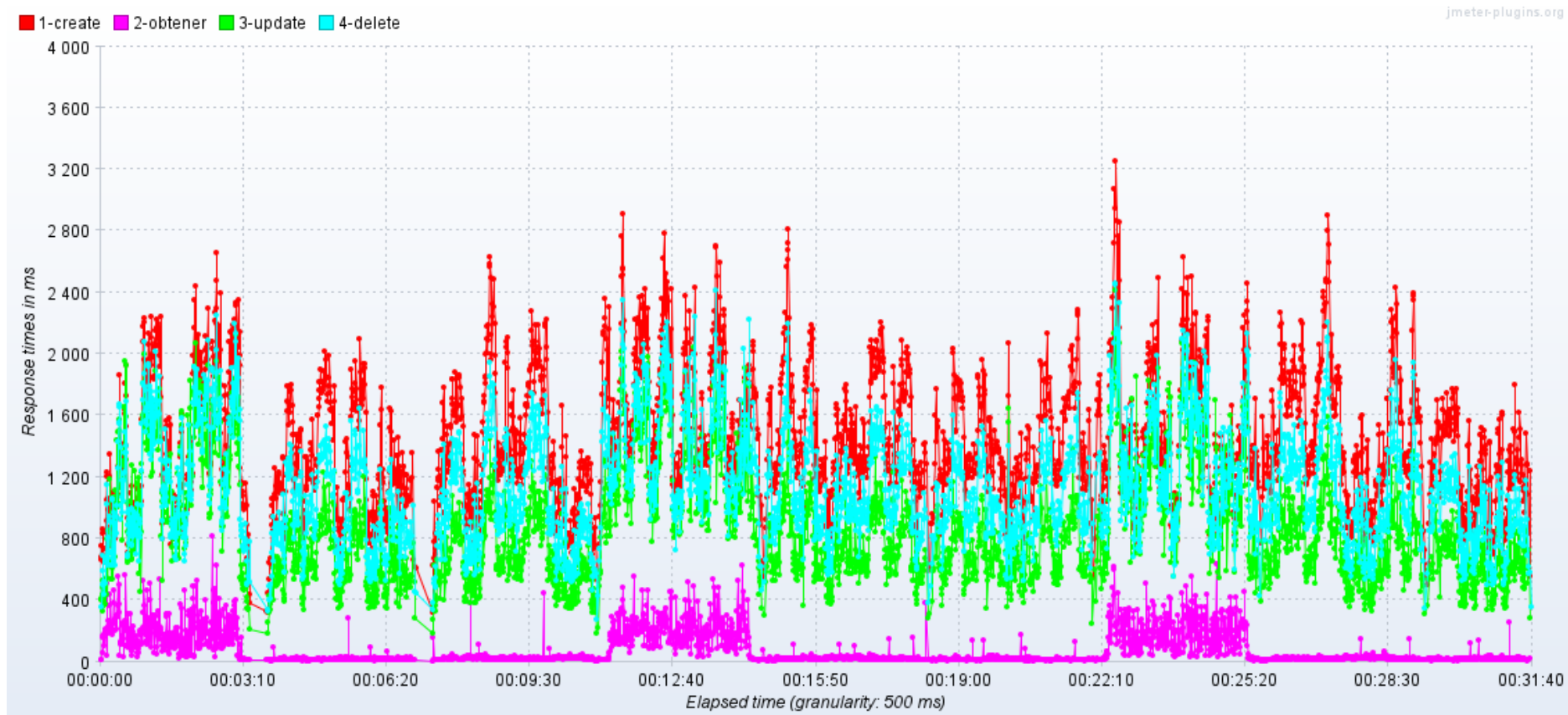
Anexo W: Resultados de throughput en la prueba de picos del aplicativo asincrónico



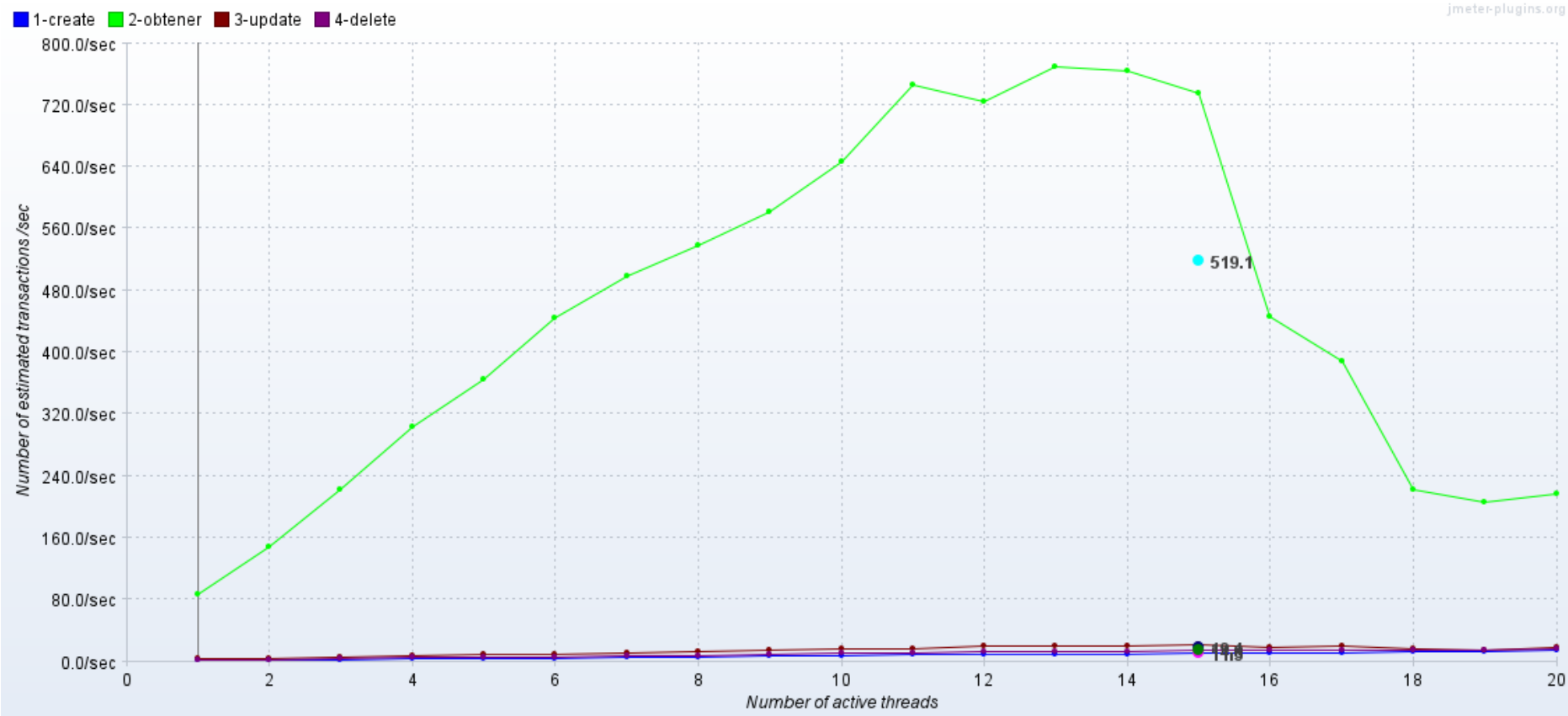
Anexo X: Resultados de tiempo de respuesta vs usuarios en la prueba de estrés del aplicativo asincrónico



Anexo Y: Resultados de tiempo de respuesta en la prueba de picos del aplicativo asincrónico



Anexo Z: Resultados de TPS vs usuarios en la prueba de picos del aplicativo asincrónico



**UNSCH**FACULTAD DE
INGENIERÍA
DE MINAS, GEOLOGÍA Y CIVIL**ACTA DE SUSTENTACIÓN DE TESIS N° 68-2026-FIMGC****PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS**

En la Universidad Nacional de San Cristóbal de Huamanga, en la ciudad de Ayacucho, en cumplimiento a la **Resolución Decanal No 375-2026-FIMGC-D**, a los **diecisiete días del mes de septiembre de 2026**, siendo las **10:10 a.m.**, reunidos en el **Auditorio de la Escuela Profesional de Ingeniería de Minas**, bajo la presidencia del **MSc. José Ernesto ESTRADA CARDENAS**, y los miembros: **Dr. Ing. Manuel Avelino LAGOS BARZOLA**, **Dr. Ing. José Elías YAURI VIDALÓN** y **Dr. Ing. Hubner JANAMPA PATILLA**, actuando como secretario docente el **Ing. Saul Walter RETAMOZO FERNANDEZ**, para proceder a la sustentación de tesis para optar el **Título Profesional de Ingeniero de Sistemas**, del Bachiller en Ingeniería de Sistemas:

LEONEL BRADONS ROJAS HUARANCCA

Quien presentó la tesis denominada:

Análisis comparativo del rendimiento en persistencia de datos entre paradigmas imperativo y reactivo, mediante aplicaciones CRUD, Ayacucho, 2025

Los señores miembros del jurado luego de expuesta la tesis y absueltas las preguntas, deliberaron y declararon:

Aprobado con 17 (diecisiete)

Siendo las **11:43 a.m.** del día **17 de septiembre del 2026**, culmina el acto de sustentación de tesis, y en conformidad de lo actuado los miembros del jurado firmamos al pie del presente.

MSc. José Ernesto ESTRADA CARDENAS
Presidente

Dr. Ing. José Elías YAURI VIDALÓN
Miembro

Dr. Ing. Hubner JANAMPA PATILLA
Miembro

Dr. Ing. Manuel Avelino LAGOS BARZOLA
Miembro - Asesor

Ing. Saul Walter RETAMOZO FERNANDEZ
Secretario docente de la FIMGC



CONSTANCIA DE ORIGINALIDAD DE TRABAJO DE INVESTIGACIÓN

CONSTANCIA N° 044-2026-KPS-FIMGC/UNSCH

El que suscribe; responsable verificador de originalidad de trabajos de tesis de pregrado con el software Turnitin, en segunda instancia para las **Escuelas Profesionales** de la **Facultad de Ingeniería de Minas, Geología y Civil**; en cumplimiento a la **Resolución de Consejo Universitario N° 039-2021-UNSCH-CU**, Reglamento de Originalidad de Trabajos de Investigación de la Universidad Nacional San Cristóbal de Huamanga y **Resolución Decanal N° 697-2024-FIMGC-D**, deja constancia de originalidad de trabajo de investigación, que el/la Sr./Srta.

Nombres y Apellidos : Leonel Bradons Rojas Huaranca
Escuela Profesional : INGENIERÍA DE SISTEMAS
Título de la Tesis : Análisis comparativo del rendimiento en persistencia de datos entre paradigmas imperativo y reactivo, mediante aplicaciones CRUD, Ayacucho, 2025
Evaluación de la Originalidad : 4% Índice de Similitud
Identificador de la entrega : 3048773496

Por tanto, según los Artículos 12, 13 y 17 del Reglamento de Originalidad de Trabajos de Investigación, es **PROCEDENTE** otorgar la **Constancia de Originalidad** para los fines que crea conveniente.

En señal de conformidad y verificación se firma la presente constancia

Ayacucho, 05 de octubre de 2026



Firmado digitalmente por:
PERALTA SOTOMAYOR Karel
FAU 20143660754 soft
Motivo: Soy el autor del documento
Fecha: 05/10/2026 21:06:54-0500

Análisis comparativo del rendimiento en persistencia de datos entre paradigmas imperativo y reactivo, mediante aplicaciones CRUD, Ayacucho, 2025

por Leonel Bradons Rojas Huarancca

Análisis comparativo del rendimiento en persistencia de datos entre paradigmas imperativo y reactivo, mediante aplicaciones CRUD, Ayacucho, 2025

INFORME DE ORIGINALIDAD

4%

INDICE DE SIMILITUD

4%

FUENTES DE INTERNET

1%

PUBLICACIONES

2%

TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	es.slideshare.net	Fuente de Internet	<1%
2	www.coursehero.com	Fuente de Internet	<1%
3	Esparza Ballón, Eliana Eliana. "El Catabolismo de Glucosinolatos Durante el Secado Postcosecha Determina la proporción Final de Amidas y Bencenoides Bioactivos en Harina de Maca (Lepidium meyenii)", Pontificia Universidad Catolica del Peru (Peru), 2023	Publicación	<1%
4	www.campustudionline.com	Fuente de Internet	<1%
5	Submitted to Universitat Oberta de Catalunya	Trabajo del estudiante	<1%
6	repositorioacademico.upc.edu.pe	Fuente de Internet	<1%
7	Submitted to Universidad Nacional de San Cristóbal de Huamanga	Trabajo del estudiante	<1%
8	sedici.unlp.edu.ar	Fuente de Internet	<1%
9	doku.pub	Fuente de Internet	<1%
10	repositorio.continental.edu.pe	Fuente de Internet	<1%
11	forum.opensearch.org	Fuente de Internet	<1%
12	repositorio.unajma.edu.pe	Fuente de Internet	<1%
13	hdl.handle.net	Fuente de Internet	<1%

14	www.dropbox.com Fuente de Internet	<1 %
15	Submitted to Instituto Superior de Artes, Ciencias y Comunicación IACC Trabajo del estudiante	<1 %
16	Submitted to Universidad TecMilenio Trabajo del estudiante	<1 %
17	issuu.com Fuente de Internet	<1 %
18	K. Siva Prasad Reddy. "Beginning Spring Boot 2", 'Springer Science and Business Media LLC', 2017 Fuente de Internet	<1 %
19	Submitted to Universidad Internacional del Ecuador Trabajo del estudiante	<1 %
20	docs.microsoft.com Fuente de Internet	<1 %
21	repositorio.utn.edu.ec Fuente de Internet	<1 %
22	dspace.ups.edu.ec Fuente de Internet	<1 %
23	repositorio.unsch.edu.pe Fuente de Internet	<1 %
24	rte.espol.edu.ec Fuente de Internet	<1 %
25	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Trabajo del estudiante	<1 %
26	Submitted to Universidad Cesar Vallejo Trabajo del estudiante	<1 %
27	itszotar.hu Fuente de Internet	<1 %
28	repositorio.uladech.edu.pe Fuente de Internet	<1 %