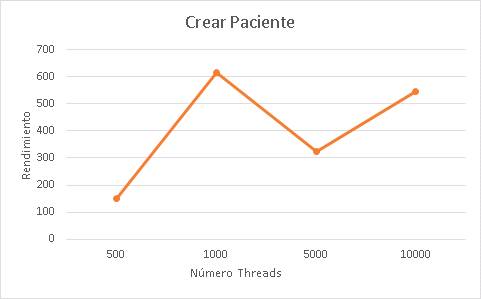
# Experimento 2 parte 1

# Se analizan y comparan los resultados de las pruebas de carga junto con los respectivos gráficos con un énfasis particular en los porcentajes de error.

## Crear paciente



|  |  |  |
| --- | --- | --- |
| Número Threads | Rendimiento | % Error |
| 500 | 148,8 | 0 |
| 1000 | 617,3 | 0 |
| 5000 | 325,3 | 2,06 |
| 10000 | 545,3 | 17,06 |

Los resultados obtenidos en esta entrega fueron muy positivos, ya que un porcentaje de error del 17%, teniendo en cuenta que los datos ingresados al sistema fueron 10.000 en un segundo, es muy manejable y mejorable. La aplicación respondió a la implementación del balanceador de carga, nivelando el “Overhead” que se presentaba previamente y disminuyendo los errores en las pruebas, dicha comparación se hará en el siguiente numeral.

## Crear Urgencia

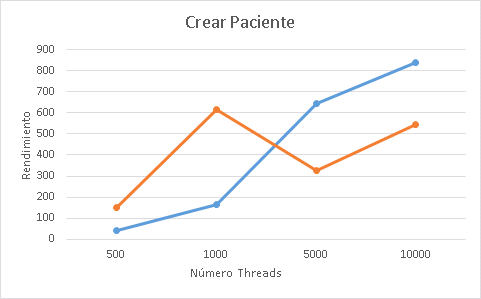
El rendimiento de las urgencias fluctúa un poco en los diferentes threads usados para implementar las pruebas de carga sobre la aplicación

Porcentajes de error:

# El porcentaje de errores presenta mejoras en ambos modelos de datos, aunque los pacientes tienen un muy buen desempeño comparado con la entrega anterior, urgencias aun presenta una mejora significativa, pero no es la que deseamos para nuestro proyecto. Los cambios, se intuye, se debieron a la reducción del overhead con el balanceador de carga, lo cual deja claro que esta arquitectura beneficia bastante nuestro proyecto.

# Se analizan y comparan los resultados de las pruebas de carga de la entrega anterior con los de la entrega actual

## Crear paciente



Donde la línea azul hace referencia a correr la aplicación sin balanceador de carga, mientras la naranja hace referencia a la inclusión del balanceador de carga.

El rendimiento aparenta ser una evidencia de que la implementación no fue la mejor, pero se debe tener en cuenta que este significa cuantos datos, en promedio, se están procesando por segundo, o la “media”. Pero el dato real que debemos emplear es el porcentaje de error evidenciado en la segunda prueba, ya que este paso de un 78% (Con 10.000 datos) a solo 17, respondiendo las necesidades de la aplicación de reducir la carga en un servidor y repartiéndola con el balanceador. Lo mismo ocurre con las urgencias, las cuales presentaron un comportamiento similar a los pacientes con la implementación del balanceador de carga, cuando se manejan entre 500 y 1000 datos, el rendimiento es creciente, cuando se trata entre 1000, 5000 y 10000 hay un ligero overhead pero la aplicación lo maneja correctamente y se recupera, para poder responder a las peticiones a tiempo y dependiendo de lo que se requiere

# Decisiones de arquitectura

Con respecto a lo propuesto en la entrega 1 del SAD, el escenario de calidad correspondiente al desempeño busca garantizar que el 98% de las peticiones de urgencia de los pacientes sean reportadas al sistema e informadas a los servicios de emergencia en menos de 10 segundos. Para garantizar esto actualmente la aplicación cuenta con:

## Patrón arquitectónico

* 1. **Arquitectura orientada a eventos (EDA):** Esta arquitectura es de gran utilidad en sistemas con necesidades de un alto grado de reacción ante cualquier evento no ordinario, lo que es muy relevante para nuestro proyecto con respecto al tema de urgencias. Ya que cuando una urgencia es reportada en el sistema lo que se espera es poder responder según lo estipula el escenario de calidad de desempeño. Esto lo logramos mediante la vista controllers del proyecto ya que cuando se genera una urgencia, se transmite al controlador de la clase Urgencia quien se encarga de unir a las demás clases para responder lo más rápido posible a la necesidad del usuario.

## Estilos arquitectónicos

* 1. **Balanceador de carga:** Favorece tácticas como la programación de recursos, múltiples copias de computación, múltiples copias de información. Lo anterior es pertinente para nuestro proyecto debido a que al contar con un balanceador de carga podremos reportar e informar a los servicios del hospital sobre la urgencia que está presentando cualquier paciente, ya que el balanceador se carga se encarga de asignar la petición al esclavo que se encuentre disponible para responder a la solicitud, por lo tanto, no es necesario esperar a que un solo servidor esté listo para responder.

## Tácticas

Con los patrones y estilos arquitectónicos expuesto anteriormente se puede afirmar que se garantizan las siguientes tácticas que fortalecen el desempeño:

* + **Reducir la sobrecarga computacional:** Al crear la clase Urgencia permitimos que cuando se genere una petición de este tipo no se tenga que acceder a diversos intermediarios para poder dar una respuesta. Si no por el contrario, está clase se encarga de orquestar en la parte lógica lo necesario para responder como se espera del sistema.
  + **Mantener múltiples copias de datos o cálculos:** Al tener diferentes esclavos que pueden responder una solitud sin tener restricciones de entradas, garantizamos que la información del hospital esté disponible en cada uno de ellos lo que permite que cualquiera pueda respaldar al sistema en el proceso de responder a una solicitud.
  + **Aumentar los recursos disponibles:** Al ser un grupo de cuatro personas podemos garantizar que la aplicación no solo pueda usar los recursos físicos de una terminal sino en este caso de 3, ya que uno de los integrantes del grupo será el balanceador. Lo anterior, permite que la respuesta a solicitudes pueda ser menor a que si solo se pudiera dar respuesta desde una terminal.
  + **Arbitraje de recursos:** Por medio de la asignación de prioridad simple se puede crearla condición para que el sistema tome las peticiones de urgencia como prioridad para ser respondidas frente a los otros tipos de solicitudes sobre el sistema.
  + **Políticas de programación Least-Conn:** Por medio de la configuración del balanceador de carga se garantiza que las peticiones serán enviadas al esclavo con más disponibilidad para responder, lo que garantiza que la urgencia se responderá con la mayor brevedad posible.

# Se evidencia un proceso basado en análisis de los requerimientos, diseño de elementos de arquitectura, diseño de la solución, implementación y pruebas. Se cuenta con artefactos que soportan el proceso y con un depósito de las fuentes.

**Implementación y pruebas, Diseño de la solución:**

Teniendo en cuenta los resultados de este Experimento, y comparándolos con resultados anteriores, se puede evidenciar que la prioridad del trabajo ha sido garantizar la rápida y correcta creación de objetos en la base de datos. Con esto en mente, hay que tener en cuenta que para este Experimento se implementó un balanceador de carga, por lo que las pruebas de carga tuvieron ligeros cambios en la eficiencia, o el número de objetos creados sin error. Aun así, se mejoró el desempeño del proyecto, logrando un porcentaje de error de 17.06% cuando mandamos 10.000 datos en un segundo, lo cual es un porcentaje de error corregible o manejable, teniendo en cuenta que antes el proyecto se ejecutaba solo en una máquina y se obtenía un porcentaje de error bastante grande, oscilando entre 78% y 92% hablando de Pacientes. El rendimiento fluctúa un poco teniendo en cuenta esta implementación, pero el proyecto responde correctamente a lo que creamos o buscamos usando JMeter. Esto se debe a que el tipo de balanceador de carga implementado usa una arquitectura Least-conn, la cual asigna las peticiones a los servidores con menor carga para garantizar que los datos no se pierdan por una sobrecarga de información repentina.

**Artefactos:**

Los artefactos utilizados son las prioridades en el desarrollo del proyecto, siendo estos las urgencias y el envío de información de los sensores. Con las pruebas realizadas, se puede ver que estos son la prioridad en el proyecto, evaluado en el pequeño porcentaje de errores cuando creamos o devolvemos objetos. Actualmente, el artefacto de medición no se ha implementado para esta entrega con este tipo de arquitectura de datos, su implementación se hará en la entrega final de este experimento.

**Diseño de elementos de arquitectura:**

La base de datos en PostgreSQL fue implementada debido a las características que benefician nuestro proyecto, entre las que se encuentran Alta concurrencia y amplia variedad de tipos nativos. Con esto en mente, se ve que la aplicación se benefició considerablemente de la base de datos ya que, aunque se hizo con un balanceador de carga, los resultados fueron mejores que la implementación del experimento anterior, y agregar datos se realiza con mejor eficiencia y menor porcentaje de error.