PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS

Sistema Distribuido de Préstamo de Libros

Segunda Entrega

Proyecto de Introducción a Sistemas Distribuidos

Período Académico 2025-30

Integrantes:

Valeria Catalina Caycedo Ramírez - v_caycedo@javeriana.edu.co
Jorge Esteban Gomez Zuluaga - gomezjo2@javeriana.edu.co
Karen Daniela Medina Naranjo - kdaniela.medina@javeriana.edu.co
Jeisson Camilo Ruiz Cristancho - jeissonc_ruizc@javeriana.edu.co

21 de noviembre de 2025

I. Introducción

Este documento presenta los resultados de la segunda entrega del proyecto, en la cual se realizaron pruebas de carga con la herramienta Locust para evaluar el comportamiento del sistema bajo diferentes condiciones de uso. Adicionalmente, se analiza en profundidad el impacto de cada modelo de comunicación en atributos clave como el tiempo de respuesta, el throughput y, especialmente, la consistencia de los datos, un aspecto crucial en un sistema de préstamo bibliotecario donde la integridad de las transacciones es indispensable. A través de este análisis, se busca ofrecer una visión clara sobre la conveniencia de cada enfoque según los requisitos específicos de las operaciones del sistema.

II. Arquitectura del Sistema

El sistema sigue una arquitectura de microservicios simplificada, basada en el patrón de Actores comunicación por paso de mensajes, diseñada para ser escalable, modular y tolerante a fallos.

II-A. Capa de Cliente (Solicitante)

- Representada por el `proceso_solicitante` (o usuarios simulados con Locust).
- Genera peticiones de operaciones: préstamo, renovación y devolución.
- Envía las solicitudes al Gestor de Carga.
- Se comunica mediante ZeroMQ usando el patrón REQ.

II-B. Capa de Orquestación (Gestor de Carga)

- El componente central `gestor_carga` actúa como Load Balancer y Router.
- Recibe todas las peticiones de los clientes.
- Distribuye la solicitud al Actor correspondiente según el tipo de operación.
- Utiliza ZeroMQ para comunicación de baja latencia.

II-C. Capa de Procesamiento (Actores)

- Servicios independientes: `actor_prestamo`, `actor_renovacion`, `actor_devolucion`.
- Cada actor encapsula una lógica de negocio específica.
- Los actores consultan o actualizan la base de datos a través del Gestor de Almacenamiento.
- Permiten escalamiento horizontal según el tipo de operación sin afectar a los demás.

II-D. Capa de Almacenamiento (Gestor de Almacenamiento + Base de Datos)

- `gestor_almacenamiento`: gestiona las conexiones y consultas hacia la base de datos.
- Cluster PostgreSQL:
  - Primario (postgres_primary): recibe todas las escrituras y lecturas críticas.
  - Réplica (postgres_replica): mantiene una copia sincronizada de los datos.
- La réplica permite alta disponibilidad (HA) y recuperación ante fallos.

II-E. Capa de Alta Disponibilidad (Failover Monitor)

- Un servicio guardián (`failover_monitor`) supervisa la salud del nodo primario.
- En caso de caída del primario:
  - Promueve la réplica a nuevo nodo maestro.
  - Orquesta el proceso de Failover Automático para garantizar continuidad del servicio.

Nota: Esta arquitectura se usa para la comunicación síncrona. La comunicación asíncrona utiliza un modelo publicador/suscriptor.

III. Especificaciones de Hardware y Software

III-A. Software

- Sistema Operativo Base: Windows 10/11 (host), Linux (contenedores).

- Plataforma de Contenedores: Docker Desktop (Engine v24+).
- Orquestación: Docker Compose.
- Lenguaje de Programación: Python 3.9+.
- Base de Datos: PostgreSQL 16.
- Middleware de Comunicación: ZeroMQ (pyzmq).
- Herramienta de Pruebas de Carga: Locust.
- Librerías Adicionales: psycopg2-binary (driver de base de datos).

Cuadro I: Especificaciones de hardware requeridas

Componente	**Especificación**
Procesador (CPU)	Intel Core i7
Memoria RAM	16 GB
Almacenamiento	1 TB
Conexión de Red	Wi-Fi

IV. Herramientas de Medición Utilizadas

Para la recolección de métricas y el análisis del comportamiento del sistema se utilizaron varias herramientas. En primer lugar, Locust sirvió como la plataforma principal de pruebas de carga distribuida, permitiendo simular usuarios concurrentes y medir tanto los tiempos de respuesta como la tasa de peticiones por segundo (RPS). Adicionalmente, se empleó `docker stats` para monitorear en tiempo real el consumo de recursos, especialmente CPU y memoria, de los contenedores. También se consultaron los registros mediante `docker logs` con el fin de verificar la correcta ejecución del proceso de *failover* y la detección de errores. Finalmente, se desarrollaron scripts personalizados en Python (como `analizar_resultados.py`) para procesar los archivos CSV generados por Locust y calcular las estadísticas finales de rendimiento.

V. Análisis de Comunicación Síncrona vs Asíncrona

En el contexto de este sistema distribuido de biblioteca, se implementan ambos paradigmas de comunicación para diferentes propósitos:

V-A. Comunicación Síncrona (Cliente-Gestor-Actores)

- Implementación: Patrón REQ/REP (Request/Reply) de ZeroMQ.
- Flujo:
  - El *proceso_solicitante* (o usuario Locust) envía una petición, por ejemplo, una renovación.
  - El cliente queda bloqueado esperando la respuesta.
  - El *gestor_carga* recibe la solicitud.
  - El Gestor realiza una llamada síncrona al Actor correspondiente (por ejemplo, *actor_renovacion*).
  - El Actor procesa la lógica, consulta la base de datos y devuelve una respuesta al Gestor.
  - El Gestor envía finalmente la respuesta al cliente.
- Garantiza consistencia inmediata para el usuario, quien sabe si su operación fue exitosa o falló en el momento.

Cuadro II: Resultados de pruebas de carga

Usuarios	Tiempo promedio (ms)	Solicitudes procesadas	Requests/s
4	6.4330	388	3.2654
4	6.0094	2	91.2191
6	6.9259	567	4.7766
10	6.7389	949	7.9669

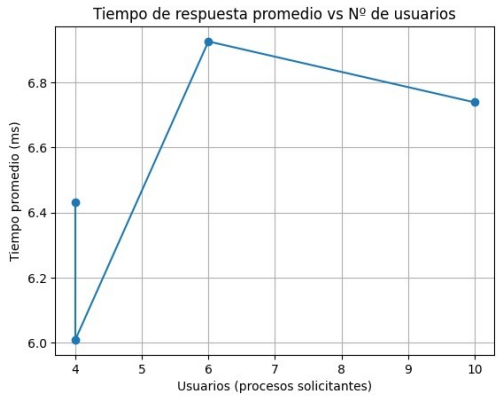

Figura 1: Gráfica del tiempo de respuesta promedio vs. el número de usuarios recibidos cada dos segundos.

La Figura 1 muestra la estabilidad temporal del sistema bajo carga creciente. Se observa que el tiempo de respuesta se mantiene consistentemente alrededor de 6-7 milisegundos, incluso cuando la carga de usuarios se incrementa de 4 a 10 usuarios concurrentes. La línea demuestra que el sistema no sufre degradación significativa en este rango de carga, mostrando una buena capacidad de procesamiento y una arquitectura balanceada que maneja eficientemente las solicitudes simultáneas.

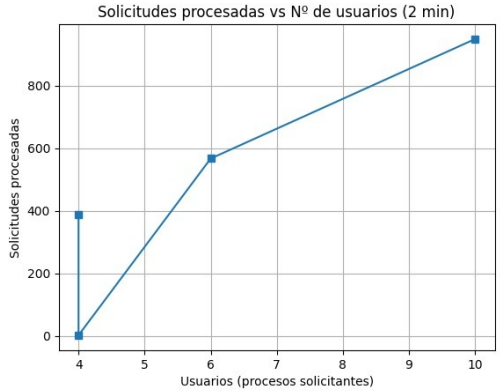

Figura 2: Gráfica de la cantidad de solicitudes procesadas respecto al número de usuarios.

La Figura 2 evidencia el comportamiento escalable del sistema en términos de throughput. A medida que el número de usuarios aumenta de 4 a 10, la cantidad total de solicitudes procesadas crece de manera casi lineal, pasando de aproximadamente 388 a 949 solicitudes. Es decir, las solicitudes procesadas por segundo también aumentan de manera proporcional al número de usuarios, alcanzando cerca de 8 req/s con 10 usuarios. Sin embargo, se observa un valor atípico en el caso de 4 usuarios (91 req/s), probablemente asociado a una operación muy rápida, posiblemente una devolución local.

V-B. Comunicación Asíncrona (Replicación de Base de Datos)

- Implementación: PostgreSQL Streaming Replication.
- Flujo:
  - Cuando se escribe un cambio en `postgres_primary`, la transacción se confirma tras escribirse en el WAL.
  - No espera confirmación de `postgres_replica`.
  - El envío de los datos ocurre en segundo plano.
- Reduce la latencia de escritura al no bloquear la transacción. Como compromiso, puede existir una breve ventana de pérdida de datos si el primario falla antes de replicar los cambios (RPO ¿0).

Cuadro III: Resultados de pruebas de carga

Usuarios	Tiempo promedio (ms)	Solicitudes procesadas	Requests/s
4	2001.5979824644144	066	0.4993482160174717
4	1.1036465591713984	377	3.1802293657078440
4	2001.6197760899860	018	0.4994222978923422
4	2001.9097114676860	067	0.4990437410792105
4	2001.6283328716568	065	0.4993191914284964
6	2001.7712729317800	070	0.4993149169005932
6	2001.8947334850536	068	0.4992872299980273
6	1.0299146280137852	569	4.7854774236840590
10	1.1602403261722662	936	7.8582138907544710
10	2001.7587776425520	079	0.4992555743403982
10	2001.7721652984620	079	0.4992786745719054

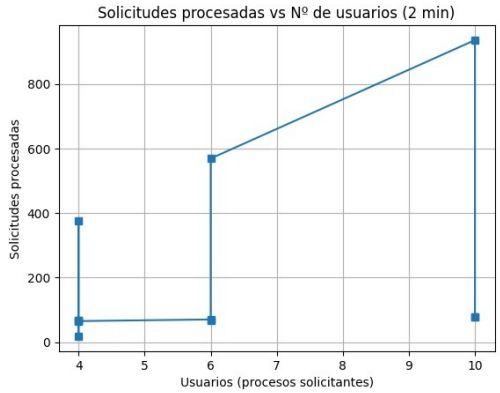

Figura 3: Gráfica de la cantidad de solicitudes procesadas respecto al número de usuarios.

La Figura 3 evidencia que el sistema asíncrono maneja solicitudes mucho más rápido cuando el procesamiento es simple, alcanzando tiempos de 1 ms y casi 8 req/s. Sin embargo, cuando se presentan solicitudes que requieren un trabajo más profundo, el throughput disminuye a valores cercanos a 0.5 req/s. Esto muestra que el sistema asíncrono presenta un comportamiento más variable y depende en gran medida del tipo de operación y de la sincronización con la base de datos.

Figura 4: Gráfica del tiempo de respuesta promedio vs. el número de usuarios recibidos cada dos segundos.

La Figura 4 presenta valores divididos en dos grupos: órdenes rápidas con tiempos cercanos a 1 ms y órdenes lentas con tiempos alrededor de 2000 ms. Este comportamiento refleja el carácter inherentemente asincrónico del sistema, donde algunas solicitudes son confirmadas de inmediato mediante un ACK, mientras que otras dependen del procesamiento interno y del acceso a la base de datos.

V-C. Análisis Comparativo

Al comparar la comunicación síncrona con la asíncrona, se observan diferencias importantes relacionadas con el rendimiento, pero especialmente con la consistencia, un atributo fundamental en este tipo sistemas como lo es el de una biblioteca. En el modelo síncrono, cada solicitud del usuario se procesa completamente antes de devolver una respuesta, lo que garantiza que el cliente reciba un resultado definitivo y coherente con el estado real de la base de datos. Esto evita estados intermedios y elimina la incertidumbre sobre si una operación fue aplicada correctamente o no. Además, los tiempos de respuesta medidos en las pruebas se mantuvieron bajos (entre 6 y 7 ms), aumentando la predictibilidad del sistema y asegurando una experiencia uniforme para el usuario.

Por otro lado, la comunicación asíncrona devuelve un mensaje de recibido sin esperar el procesamiento final. Aunque este permite tiempos muy rápidos en operaciones simples (alrededor de 1 ms) y puede incrementar el throughput, introduce una variabilidad muy marcada: algunas solicitudes pueden tardar más de 2000 ms en completarse. Esto genera un comportamiento bimodal y difícil de anticipar. Desde la perspectiva de consistencia, el modelo asíncrono es impreciso, ya que la respuesta inicial no representa necesariamente el estado real del sistema, porque las operaciones pueden escribirse después en la base de datos, con un riesgo adicional cuando la replicación también es asíncrona. Esto puede llevar a que los usuarios trabajen con información desactualizada o que solicitudes conflictivas se procesen fuera de orden, afectando la integridad del sistema.

En resumen, aunque la asincronía ofrece flexibilidad y potencialmente mejor rendimiento en ciertos escenarios, sacrifica la consistencia inmediata. El enfoque síncrono, en cambio, garantiza resultados coherentes y transacciones completamente verificadas antes de responder al usuario.

VI. Conclusiones

La elección entre comunicación síncrona y asíncrona no implica que una sea estrictamente mejor que la otra, sino que se trata de una decisión arquitectónica que debe alinearse con los requisitos de cada componente del sistema. Para operaciones críticas de negocio en la biblioteca, como préstamos, renovaciones y devoluciones, donde la integridad transaccional y la consistencia son importantes, la comunicación síncrona es la opción más adecuada, ya que garantiza que cada respuesta entregada al usuario refleje el estado definitivo del sistema, evitando discrepancias y reduciendo la posibilidad de errores por estados intermedios o concurrencia. Por otro lado, la comunicación asíncrona resulta útil en la infraestructura de replicación de la base de datos y mecanismos de alta disponibilidad, ofreciendo un balance óptimo entre rendimiento, escalabilidad y recuperación ante fallos, aunque su consistencia eventual y su variabilidad temporal la hacen menos apropiada para operaciones críticas. La efectividad del sistema distribuido de biblioteca esta en esta que de cierta forma se complementen, donde ambos enfoques coexistan para ofrecer un servicio que asegura simultáneamente la consistencia para los usuarios finales y la resiliencia de la infraestructura subyacente.

Referencias

[1] ZeroMQ, “ZeroMQ: Python Language Bindings,” [Online]. Available: https://zeromq.org/languages/python/. [Accessed: 21-Nov-2025].

[2] PostgreSQL Global Development Group, “PostgreSQL Documentation,” [Online]. Available: https://www.postgresql.org/docs/. [Accessed: 21-Nov-2025].

[3] Locust, “Locust Documentation,” [Online]. Available: https://docs.locust.io/en/stable/. [Accessed: 21-Nov-2025].

[4] H. A. Viloria Matheus and J. Hamburger, “Uso de las herramientas comunicativas en los entornos virtuales de aprendizaje / Use of communicative tools in virtual learning environments ,” *Chasqui. Revista Latinoamericana de Comunicación*, no. 140, pp. 367–384, Apr.-Jul. 2019. [Online]. Available: https://share.google/zquMdXKhqomb9q9rm. [Accessed: 21-Nov-2025].