

SEGUNDA ENTREGA ENTREGA PROYECTO



Sistema Distribuido de Préstamo de Libros

Arley Bernal Muñetón
Kevin Garay Gaitán
Juan Esteban Bello

Introducción a Sistemas Distribuidos

20 de Noviembre de 2025

2025-30

1. Introducción

El Sistema Distribuido de préstamo de Libros para la Universidad Ada Lovelace es una solución diseñada para gestionar operaciones de préstamo, renovación y devolución de libros en múltiples sedes bibliotecarias. El sistema implementa una arquitectura distribuida que garantiza alta disponibilidad, tolerancia a fallos y escalabilidad, utilizando patrones de comunicación síncronos y asíncronos mediante la librería ZeroMQ.

1.1 Objetivos específicos

- Gestionar préstamos, renovaciones y devoluciones de libros en dos sedes físicas.
- Implementar replicación asíncrona de base de datos para consistencia eventual.
- Proveer mecanismos de tolerancia a fallos automáticos.
- Garantizar alto desempeño bajo diferentes cargas de trabajo.
- Evaluar el impacto de diferentes estrategias de comunicación en el rendimiento.

2. Modelos del Sistemas

2.1 Modelo Arquitectónico

El sistema sigue una arquitectura híbrida cliente-servidor con elementos de microservicios, distribuida físicamente en tres capas principales.

- **Capa de Presentación:** Procesos Solicitantes (PS) que interactúan con usuarios finales.
- **Capa de Lógica de Negocio:** Gestor de Carga (GC) que coordina las operaciones.
- **Capa de Persistencia:** Gestor de Almacenamiento (GA) con réplicas de base de datos.

Aplicación al proyecto:

1. Arquitectura multi-sede: Cada sede opera semi-independientemente con su propio GC y GA.
2. Comunicación descentralizada: Los componentes se comunican directamente sin intermediarios centralizados.
3. Réplicas maestro-maestro: Ambas sedes mantienen réplicas que se sincronizan asíncronamente.
4. Failover automático: Detección y recuperación transparente ante fallas.

2.2 Modelo de Interacción

El modelo de interacción implementa patrones mixtos según el tipo de operación, optimizando para consistencia vs rendimiento:

Patrón Request-Replay (Síncrono):

- Operaciones de préstamo de libros (requieren consistencia inmediata)
- Comunicación PS → GC → Actor → GA → Respuesta

Patrón Publicador/Suscriptor (Asíncrono):

- Operaciones de devolución y renovación (toleran consistencia eventual)
- GC publica tópicos, Actores suscritos procesan asíncronamente

Aplicación al proyecto:

- Préstamos: Comunicación síncrona para garantizar que el usuario reciba confirmación inmediata de disponibilidad.
- Devoluciones/Renovaciones: Comunicación asíncrona para mejor rendimiento, con confirmación inmediata al usuario.
- Actualización de réplicas: Sincronización asíncrona entre sedes para mantener consistencia eventual.

2.3 Modelo de Fallos

El sistema implementa un modelo de tolerancia a fallos basado en replicación y recuperación automática:

Detección de Fallas:

- Health checks periódicos entre GC y GA mediante heartbeat.
- Timeouts configurados en comunicaciones síncronas (5 segundos).
- Monitoreo continuo del estado de los componentes.

Recuperación Automática:

- Failover automático a réplica secundaria cuando falla GA primario.
- Reconexión transparente para clientes sin interrupción del servicio.

- Resincronización de datos cuando el componente recupera operación.
- Cola de mensajes pendientes durante periodos de indisponibilidad.

Aplicación al proyecto:

- Failover de GA: Si falla el GA de una sede, automáticamente se usa la réplica de la otra sede.
- Retry con backoff exponencial: Reintentos automáticos en comunicaciones fallidas.
- Persistencia de estado: Recuperación de operaciones pendientes después de fallas
- Transparencia de fallas: Los PS no perciben interrupciones en el servicio.

2.4 Modelo de Seguridad

Aunque el enfoque principal es la distribución y tolerancia a fallos, se implementan medidas básicas de seguridad:

Control de Acceso:

- Validación de formatos de mensajes entre componentes
- Verificación de integridad de datos en operaciones críticas
- Autenticación básica mediante tokens en comunicaciones

Protección de Datos:

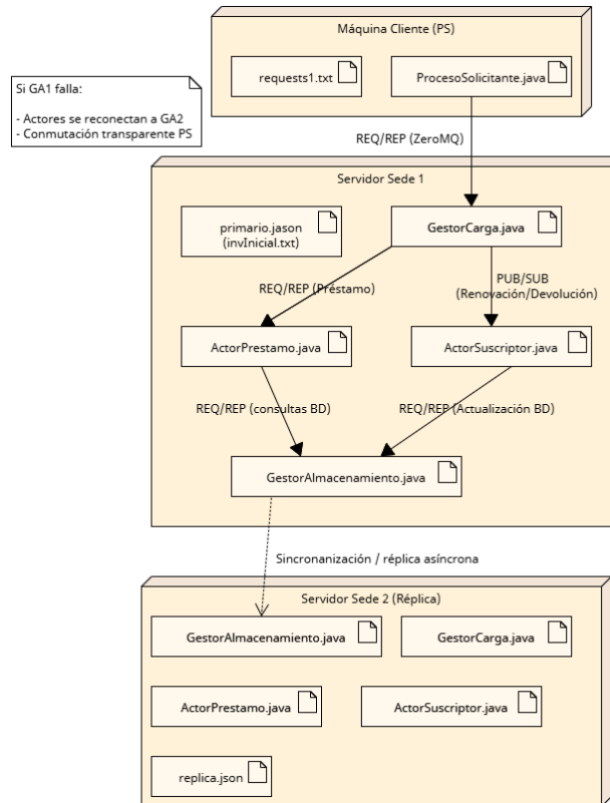
- Validación de reglas de negocio (máximo 2 renovaciones por libro)
- Consistencia en actualizaciones de inventario
- Prevención de condiciones de carrera en operaciones concurrentes

Aplicación al proyecto:

1. Validación de operaciones: Verificación de existencia y disponibilidad de libros antes de procesar.
2. Control de consistencia: Prevención de préstamos duplicados o inconsistentes.
3. Auditoría: Registro de todas las operaciones para trazabilidad y diagnóstico
4. Validación de estado: Verificación de que los libros pueden ser renovados según las políticas

3 Modelos del Sistemas

3.1 Diagrama de Despliegue



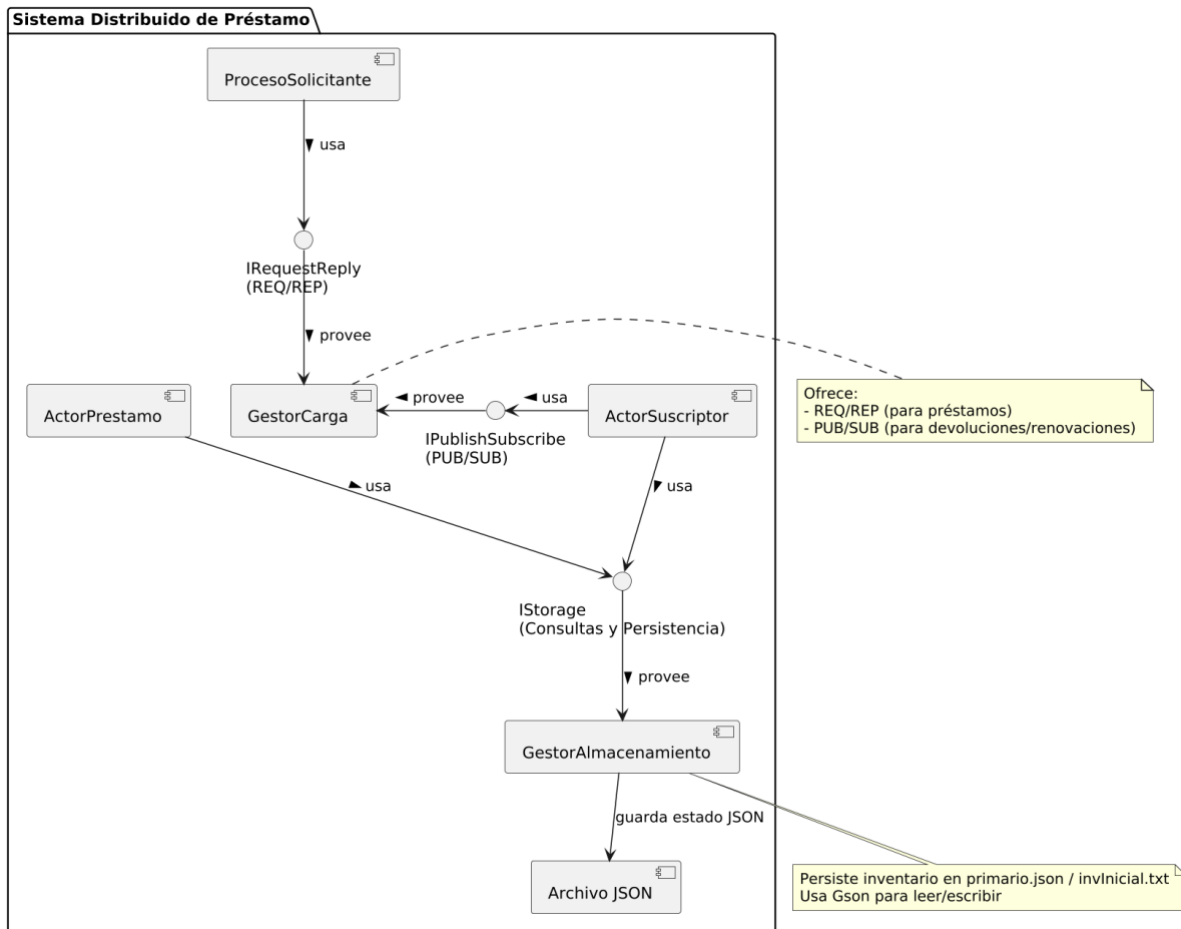
Descripción del despliegue:

- **Máquina Cliente (PS):** Ejecuta ProcesoSolicitante.java con archivo requests1.txt
- **Servidor Sede 1:** Contiene GestorCarga.java, ActorPrestamo.java, ActorSuscriptor.java y GestorAlmacenamiento.java con primario.json
- **Servidor Sede 2 (Réplica):** Contiene componentes equivalentes con replica.json
- **Mecanismo de Failover:** Si GA1 falla, actores se reconectan automáticamente a GA2

Patrones de comunicación implementados:

- **REQ/REP (ZeroMQ):** Para operaciones de préstamo (síncronas) que requieren respuesta inmediata.
- **PUB/SUB (ZeroMQ):** Para operaciones de renovación/devolución (asíncronas) que toleran procesamiento diferido.
- **Sincronización asíncrona:** Entre primario.json y replica.json mantener consistencia eventual.

3.2 Diagrama de Componentes



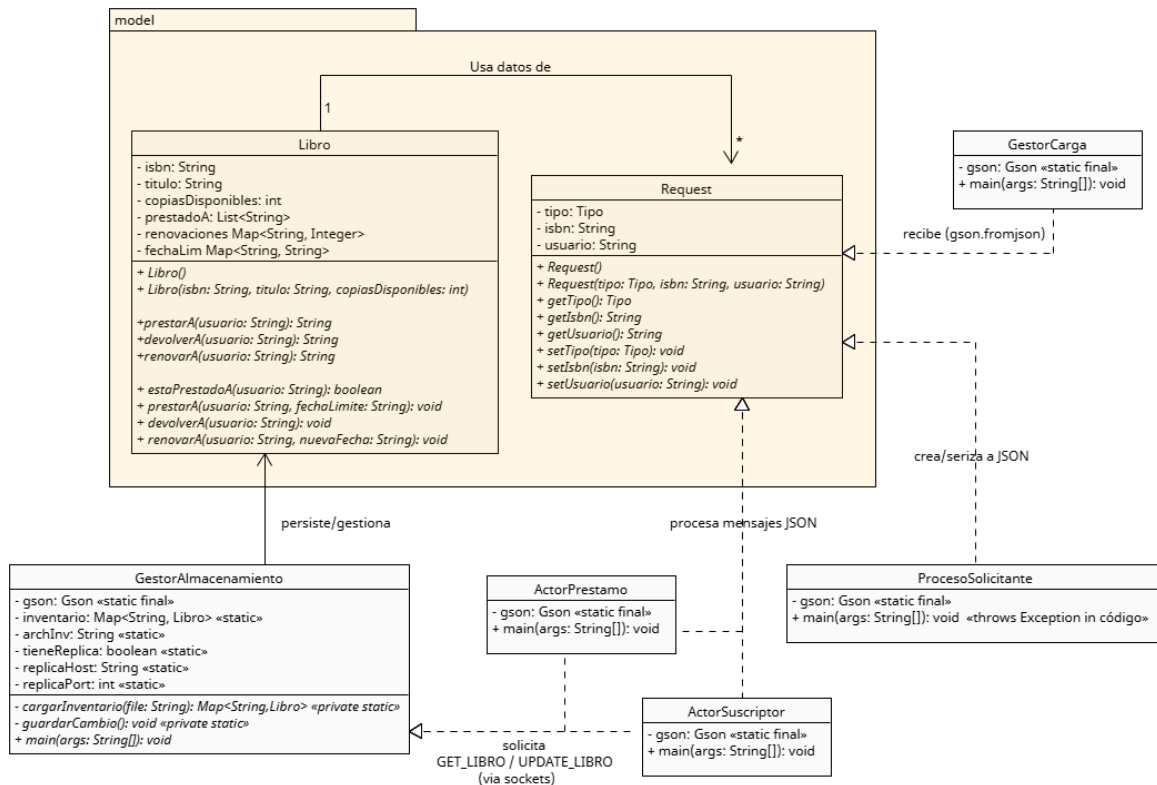
Descripción de componentes:

- **Proceso Solicitante:** Genera solicitudes usando patrones REQ/REP y PUB/SUB.
- **Gestor Carga:** Coordina operaciones y provee interfaces para Actores.
- **Actor Préstamo:** Maneja solicitudes síncronas de préstamo.
- **Actor Suscriptor:** Maneja solicitudes asíncronas de devolución/renovación.
- **Gestor Almacenamiento:** Implementa interfaz IStorage para persistencia JSON.

Interfaces y contratos definidos:

- RequestReply (REQ/REP): Contrato síncrono para operaciones que requieren respuesta inmediata.
- PublishSubscribe (PUB/SUB): Contrato asíncrono para operaciones de procesamiento diferido.
- IStorage: Interfaz de persistencia abstracta para operaciones de base de datos.

3.3 Diagrama de Clases



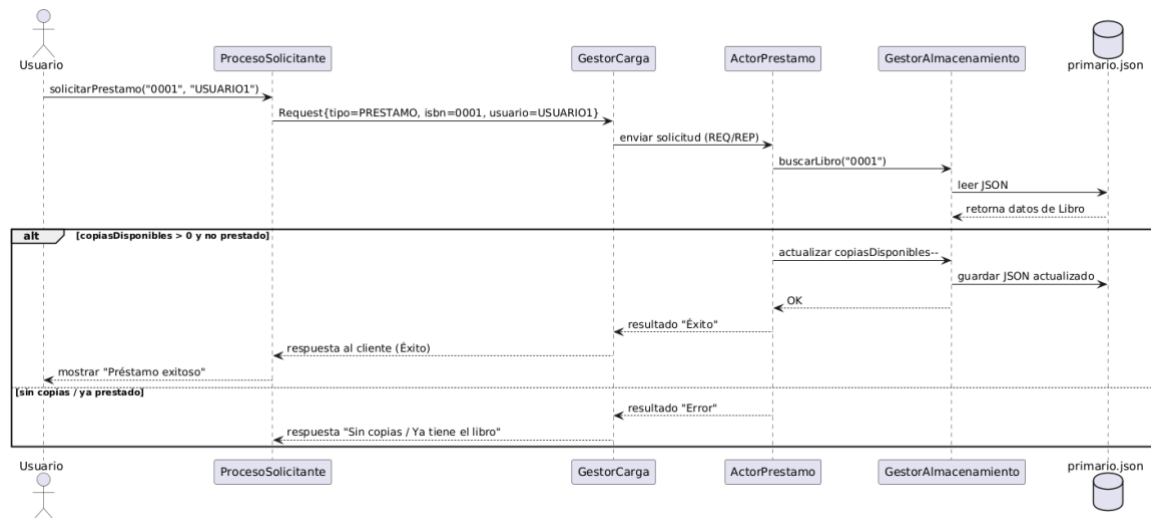
Descripción del modelo de datos:

- Clase Libro:
 - Atributos: isbn, titulo, copiasDisponibles, prestadoA, renovaciones, fechaLim
 - Métodos: prestarA(), devolverA(), renovarA(), estaPrestadoA()
- Clase Request:
 - Atributos: tipo, isbn, usuario
 - Métodos: getters/setters para serialización JSON

Componentes principales:

- GestorCarga: Recibe y procesa mensajes JSON entrantes. Coordina el flujo completo de las operaciones.
- GestorAlmacenamiento: Gestiona inventario Map<String, Libro> y replicación.
- ActorPrestamo/ActorSuscriptor: Procesan solicitudes específicas.
- ProcesoSolicitante: Genera carga de trabajo automatizada y soporta ejecución concurrente para pruebas de carga.

3.4 Diagrama de Secuencia



Flujo de operación de préstamo:

1. Inicio de Operación:
 - Usuario invoca: solicitarPrestamo("0001", "USUARIO1")
 - ProcesoSolicitante crea: Request(tipo=PRESTAMO, isbn=0001, usuario=USUARIO1)
2. Comunicación Síncrona:
 - PS → GC: Envía solicitud serializada como JSON via REQ/REP
 - GC → ActorPrestamo: Delega procesamiento específico via REQ/REP
 - ActorPrestamo → GA: buscarLibro("0001") via sockets
3. Acceso a Persistencia:
 - GA → primary.json: Lectura del archivo JSON
 - Perseo y búsqueda del libro con ISBN "0001"
 - Retorno de datos del libro a memoria
4. Validación de Negocio:
 - Condición principal: [copiasDisponibles > 0 y no prestado]
 - Alternativa: [Sin copias disponibles o ya prestado]
5. Procesamiento de Respuesta:
 - Caso éxito: Actualización de estado y respuesta "Préstamo exitoso"
 - Caso error: Respuesta "Sin copias / Ya tiene el libro" sin cambios de estado
6. Propagación de Respuesta:
 - GA → ActorPrestamo: Resultado de la operación
 - ActorPrestamo → GC: Respuesta del procesamiento
 - GC → PS: Confirmación final al usuario

4 Protocolo de Pruebas

4.1 Estrategia General de Pruebas

Pruebas Unitarias (Validación de Componentes):

- Validación de formatos de mensajes JSON entre componentes.
- Lógica de negocio (reglas de renovación, límites de préstamos).
- Comportamiento de clases de dominio (Libro, Request).
- Serialización/deserialización con Gson.

Pruebas de Integración (Comunicación entre Componentes):

- Comunicación PS-GC-Actores-GA en entorno distribuido.
- Sincronización entre réplicas de BD (primario.json ↔ secundario.json).
- Patrones Pub/Sub y Request-Reply sobre ZeroMQ.
- Manejo de conexiones de red y reconexión automática

Pruebas de Sistema (Flujos Completos):

- Flujos end-to-end de todas las operaciones (préstamo, devolución, renovación).
- Comportamiento bajo carga con múltiples PS concurrentes.
- Recuperación ante fallas de componentes individuales.
- Consistencia de datos después de operaciones concurrentes.

Pruebas de Aceptación (Cumplimiento de Requerimientos):

- Verificación de cumplimiento de todos los requerimientos funcionales.
- Experiencia de usuario final en términos de respuestas y tiempos.
- Validación de políticas de negocio (límites de renovación).

4.2 Pruebas de Desempeño (Énfasis Principal)

El objetivo principal es evaluar escalabilidad y rendimiento bajo diferentes cargas, comparando arquitectura original vs modificada según opción B.

Metodología de Pruebas de Desempeño:

1. Escenarios de prueba definidos (Tabla 1):
 - Caso 1: 4 PS simultáneos generando requerimientos distribuidos equitativamente.
 - Caso 2: 6 PS simultáneos generando requerimientos distribuidos equitativamente.
 - Caso 3: 10 PS simultáneos generando requerimientos distribuidos equitativamente.

2. Variables a medir en cada escenario:

- Tiempo de respuesta promedio: Medición del latency desde envío hasta recepción de respuesta.
- Desviación estándar del tiempo de respuesta: Indicador de consistencia del performance.
- Cantidad de solicitudes procesadas: Throughput del sistema en ventana de 2 minutos.
- Solicitudes por PS: Distribución de carga entre procesos solicitantes.

3. Configuración de Entorno de Pruebas:

- Hardware: 3 máquinas virtuales con 2CPU, 4GB RAM cada una
- Software: Ubuntu 20.04, Java 8, Maven 3.6, ZeroMQ 4.3.4
- Red: Configuración LAN con latencia <1ms entre máquinas
- Datos: 1000 libros iniciales, 200 prestados (50 sede1, 150 sede2)

4. Proceso de Ejecución de Pruebas:

- Inicialización del sistema con datos de prueba
- Ejecución de benchmark automático con script run_benchmark.sh
- Captura de métricas en archivo resultados_benchmark.csv
- Análisis estadístico de resultados
- Generación de gráficos comparativos

	Experimento I	Experimento II
Numero de procesos PS generando requerimientos (distribuirlos equitativamente por sede)	Variables a medir: <ul style="list-style-type: none"> - T. Respuesta promedio (y desv. standard) de las solicitudes de préstamo. - Cantidad de solicitudes procesadas (respuesta dada a los PS) en un periodo de 2min. 	Variables a medir: <ul style="list-style-type: none"> - T. Respuesta promedio (y desv. standard) de las solicitudes de préstamo. - Cantidad de solicitudes procesadas (respuesta dada a los PS) en un periodo de 2 min.
4		
6		
10		

Tabla 1: Escenarios de prueba y variables a medir.

5 Métricas de Desempeño

5.1 Estrategia de Medición y Herramientas

Enfoque de Implementación: Inserción de Instrucciones en el Código

Para la obtención de las métricas de desempeño especificadas en la Tabla 1, se implementó una estrategia basada en la inserción de instrucciones especiales en el código en lugar de utilizar herramientas de monitoreo externas. Esta decisión se fundamenta en las siguientes ventajas:

Ventajas del Enfoque Implementado:

1. **Máxima Precisión y Control:** Las mediciones se realizan directamente en los puntos críticos del flujo de procesamiento, eliminando el overhead de herramientas externas.
2. **Integración Nativa con la Lógica de Negocio:** Las métricas se capturan como parte integral del procesamiento de solicitudes, garantizando coherencia temporal.
3. **Personalización Específica:** Las métricas se adaptan exactamente a los requerimientos del proyecto, midiendo únicamente lo necesario.
4. **Bajo Overhead:** Al evitar herramientas complejas de monitoreo, se minimiza el impacto en el rendimiento del sistema.

Arquitectura de Medición Implementada

Componentes de Medición:

1. **PSRend.java - Proceso Solicitante Instrumentado:**
 - Timestamp de Inicio: `long t0 = System.nanoTime()` antes de enviar cada solicitud.
 - Timestamp de Fin: `long t1 = System.nanoTime()` después de recibir cada respuesta.
 - Cálculo de Latencia: `long deltaMs = (t1 - t0) / 1_000_000`.
 - Contador de Throughput: Incremento de variable por cada solicitud completada.

```
28 /**
29  * Proceso Solicitante (PS): lee archivo de peticiones y las envía al Gestor de Carga (GC).
30  * Uso: java ProcesoSolicitante <requestFile> <gchost> <gchport>
31  */
32 public class PSRend {
33     private static final Gson gson = new Gson();
34
35     @Run[Debug]
36     public static void main(String[] args) throws Exception {
37         if (args.length < 3) {
38             System.out.println("Uso: java ProcesoSolicitante <archivoRequests> <gchost> <gchport>");
39             System.exit(status:1);
40         }
41
42         String file = args[0];
43         String gchost = args[1];
44         int gchport = Integer.parseInt(args[2]);
45         String psid = UUID.randomUUID().toString().substring(beginIndex:0, endIndex:4); // ID corto del PS
46
47         List<Long> tiempos = new ArrayList<>();
48         int procesadas = 0;
49         long inicio = System.currentTimeMillis();
50         long duracion = 120_000; // 2 minutos (en ms)
51
52         try (ZContext ctx = new ZContext()) {
53             ZMQ.Socket req = ctx.createSocket(SocketType.REQ);
54             req.connect("tcp://" + gchost + ":" + gchport);
55             System.out.println("PS [" + psid + "] conectado a GC en " + gchost + ":" + gchport);
56         }
```

2. AnalizadorTiempos.java - Procesador de Métricas de Tiempo:

- Agregación: Consolida todos los archivos times_ps_*.csv
- Cálculo de Promedio:
tiempos.stream().mapToLong(Long::longValue).average()
- Desviación Estándar: Cálculo matemático de varianza y raíz cuadrada
- Formato de Salida: Clave=valor para fácil procesamiento

```
13 public class AnalizadorTiempos {  
14     Run|Debug  
15     public static void main(String[] args) throws IOException {  
16         List<Long> tiempos = new ArrayList<>();  
17  
18         try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get(first:"."), glob:"times_ps_*.csv")) {  
19             for (Path p : stream) {  
20                 try {  
21                     Files.lines(p).map(Long::parseLong).forEach(tiempos::add);  
22                 } catch (Exception ignored) {}  
23             }  
24  
25             if (tiempos.isEmpty()) {  
26                 System.out.println(x:"SolicitudesMedidas=0");  
27                 System.out.println(x:"TiempoPromedio=0");  
28                 System.out.println(x:"DesviacionEstandar=0");  
29                 return;  
30             }  
31  
32             double promedio = tiempos.stream().mapToLong(Long::longValue).average().orElse(0);  
33             double varianza = tiempos.stream()  
34                 .mapToDouble(t -> Math.pow(t - promedio, 2))  
35                 .average().orElse(0);  
36             double desv = Math.sqrt(varianza);  
37  
38             // Formato limpio: clave=valor  
39             System.out.println("SolicitudesMedidas=" + tiempos.size());  
40             System.out.println("TiempoPromedio=" + String.format(format:"%.2f", promedio));  
41             System.out.println("DesviacionEstandar=" + String.format(format:"%.2f", desv));  
42         }  
43     }
```

3. AnalizadorConteo.java - Procesador de Métricas de Throughput:

- Sumatoria Total: Agrega conteos de todos los count_ps_*.txt
- Cálculo de Promedio por PS: total / (double) archivos
- Estadísticas de Distribución: Identifica carga desbalanceada

```
12 public class AnalizadorConteo {  
13     Run|Debug  
14     public static void main(String[] args) throws IOException {  
15         int total = 0;  
16         int archivos = 0;  
17  
18         try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get(first:"."), glob:"count_ps_*.txt")) {  
19             for (Path p : stream) {  
20                 try {  
21                     int n = Integer.parseInt(Files.readString(p).trim());  
22                     total += n;  
23                     archivos++;  
24                 } catch (Exception ignored) {}  
25             }  
26  
27             if (archivos == 0) {  
28                 System.out.println(x:"ArchivosEncontrados=0");  
29                 System.out.println(x:"SolicitudesTotales=0");  
30                 System.out.println(x:"PromedioPorPS=0");  
31                 return;  
32             }  
33  
34             double promedioPorPS = total / (double) archivos;  
35  
36             // Formato limpio: claves=valor  
37             System.out.println("ArchivosEncontrados=" + archivos);  
38             System.out.println("SolicitudesTotales=" + total);  
39             System.out.println("PromedioPorPS=" + String.format(format:"%.2f", promedioPorPS));  
40         }  
41     }
```

Métricas Específicas y su Implementación

1. Tiempo de Respuesta Promedio (Latency)

Propósito: Medir la capacidad de respuesta del sistema desde la perspectiva del usuario final.

```
77      long t0 = System.nanoTime();
78      req.send(json);
79      String resp = req.recvStr(flags:0);
80      long t1 = System.nanoTime();
81
82      long deltaMs = (t1 - t0) / 1_000_000;
83      tiempos.add(deltaMs);
```

2. Desviación Estándar del Tiempo de Respuesta

Propósito: Evaluar la consistencia del rendimiento e identificar problemas de variabilidad.

```
32      double promedio = tiempos.stream().mapToLong(Long::longValue).average().orElse(0);
33      double varianza = tiempos.stream()
34          .mapToDouble(t -> Math.pow(t - promedio, 2))
35          .average().orElse(0);
36      double desv = Math.sqrt(varianza);
37
```

Interpretación:

- Valor Bajo: Comportamiento predecible y consistente
- Valor Alto: Inestabilidad que afecta la experiencia de usuario

3. Cantidad de Solicitudes Procesadas (Throughput)

Propósito: Medir la capacidad del sistema para manejar carga de trabajo en un período determinado.

Protocolo de Ejecución de Mediciones

Script de Automatización: run_benchmark.sh

Funcionamiento:

- Configuración Inicial: Define parámetros de prueba (host, puerto, duración).
- Ejecución Escalonada: Pruebas con 4, 6 y 10 PS concurrentes.
- Recolección Automática: Ejecución de analizadores después de cada escenario.
- Consolidación de Resultados: Generación de resultados_benchmark.csv.

Características Clave:

- Aislamiento de Pruebas: Limpieza de archivos previos antes de cada ejecución.
- Tiempo Controlado: Ejecución exacta de 2 minutos por escenario.
- Tolerancia a Fallos: Mecanismos de espera para finalización ordenada.
- Formato Estructurado: Salida en CSV para análisis posterior.

```
run_benchmark.sh
9 # =====
10
11 GC_HOST="localhost"
12 GC_PORT=5555
13 ARCHIVO_REQUEST="src/main/resources/requests2.txt"
14 DURACION=120
15 ARCHIVO_RESULT="resultados_benchmark.csv"
16
17 echo "===== BENCHMARK COMPLETO ====="
18
19 # Encabezado CSV
20 echo "PS Simultaneos,Tiempo Promedio (ms),Desv. Estandar (ms),Solicitudes Totales,Promedio por PS" > $ARCHIVO_RESULT
21
22 # Iterar sobre los escenarios
23 for PS_COUNT in 4 6 10; do
24     echo ""
25     echo "Ejecutando pruebas con $PS_COUNT PS simultáneos..."
26     rm -f times_ps_*.csv count_ps_*.txt ps_*.log
27
28     # Lanzar PS simultáneos
29     for i in $(seq 1 $PS_COUNT); do
30         mvn -q exec:java -Dexec.mainClass=sd.library.PruebasRend.PSRend -Dexec.args="$ARCHIVO_REQUEST $GC_HOST $GC_PORT" &
31     done
32
33     echo "Esperando $DURACION segundos..."
34     sleep $DURACION
35
36     echo "Esperando que los PS terminen por sí solos..."
37     sleep 5
38     #pkill -f PSRend
39
40     echo "Esperando que los PS terminen de escribir resultados..."
41     sleep 10 # Espera 10 segundos adicionales
42
43 done
```

Validación y Calibración del Sistema de Medición

Precisión Temporal:

- Uso de System.nanoTime() para máxima resolución temporal.
- Conversión cuidadosa de nanosegundos a milisegundos.
- Consideración del overhead de medición en los resultados.

Consistencia de Datos:

- Persistencia inmediata de resultados en archivos.
- Mecanismos de sincronización para evitar corrupción.
- Validación de integridad de archivos antes del procesamiento.

Escalabilidad del Mecanismo de Medición:

- Almacenamiento en memoria durante ejecución
- Escritura asíncrona al sistema de archivos
- Formato eficiente (CSV) para minimizar I/O

Ventajas sobre Herramientas Externas

Comparación con Alternativas:

1. vs JMeter/Tools Externas:

- Menor Overhead: No requiere proceso adicional de monitoreo.
- Integración Perfecta: Las métricas son parte del flujo natural.
- Contexto Específico: Entiende exactamente la semántica de las operaciones.

2. APM (Application Performance Monitoring).

- Costo Cero: Sin licencias o configuración compleja.
- Personalización Total: Métricas adaptadas al dominio específico.
- Simplicidad: Fácil depuración y entendimiento.

Limitaciones y Consideraciones

Aspectos a Considerar:

- Overhead de Medición: Aunque mínimo, existe un impacto medible.
- Puntos Ciegos: No se miden métricas internas de componentes específicos.
- Escalabilidad: El mecanismo actual puede requerir ajustes para cargas masivas.

Mejoras Futuras Potenciales:

- Métricas Adicionales: Uso de memoria, carga de CPU, estadísticas de red.
- Visualización en Tiempo Real: Dashboard para monitoreo durante ejecución.
- Alertas Automáticas: Detección de degradación de rendimiento.

Esta estrategia de medición proporciona una base sólida y confiable para la evaluación comparativa entre la arquitectura original y la modificada, cumpliendo con todos los requerimientos especificados en la Tabla 1 mientras mantiene la simplicidad y efectividad.

6 Implementación Adicional

6.1 Operaciones Distribuidas Implementadas

Estado Actual de Implementación:

1. Comunicación Asíncrona PUB/SUB:

- Devolución y renovación completamente implementadas.
- Múltiples Actores suscritos a tópicos específicos.
- Confirmación inmediata al usuario con procesamiento diferido.
- Actualización asíncrona de base de datos después de confirmación.

2. Comunicación Síncrona REQ/REP:

- Préstamos implementados con validación en tiempo real.
- Respuesta consistente e inmediata al usuario.
- Actualización atómica del estado del libro.

3. Distribución Física en Máquinas Múltiples:

- Computador A: GC Sede 1 + Actor Devolución + Actor Renovación + GA Primario.
- Computador B: GC Sede 2 + Actor Préstamo + GA Réplica + Generador de Carga.
- Computador C: Múltiples PS para pruebas de carga distribuida

4. Mecanismo de Failover Automático:

- Detección automática de caída de GA primario.
- Reconexión transparente a GA réplica.
- Continuidad de operaciones sin intervención manual

6.2 Mecanismo de Generación de Requerimientos

Implementación Principal: Generación desde Archivo de Texto

Formato del Archivo de Entrada (requests1.txt):

```
1 # Formato por línea: Tipo, ISBN, Usuario
2 # TIPO = PRESTAMO | DEVOLUCION | RENOVACION
3
4 #Éxito
5 PRESTAMO,0001,USUARIO1
6 #Éxito
7 PRESTAMO,0001,USUARIO2
8 #Éxito (quedan 0 copias)
9 PRESTAMO,0001,USUARIO3
10 #Error: ya tiene el libro
11 PRESTAMO,0001,USUARIO1
12 #Error: sin copias disponibles
13 PRESTAMO,0004,USUARIO9
14
15 #Error: máximo de renovaciones alcanzado
16 RENOVACION,0003,USUARIO5
17 #Éxito
18 RENOVACION,0001,USUARIO2
19 #Error: no tiene el libro
20 RENOVACION,0002,USUARIO4
21
22 #Éxito
23 DEVOLUCION,0001,USUARIO3
24 #Error: no tiene el libro
25 DEVOLUCION,0002,USUARIO4
```

```
1 # Archivo con peticiones de ejemplo (mínimo 20)
2 # Formato por línea: Tipo, ISBN, Usuario
3 # TIPO = PRESTAMO | DEVOLUCION | RENOVACION
4 PRESTAMO,0001,USUARIO1
5 PRESTAMO,0002,USUARIO2
6 PRESTAMO,0003,USUARIO3
7 PRESTAMO,0004,USUARIO4
8 RENOVACION,0001,USUARIO1
9 DEVOLUCION,0002,USUARIO2
10 RENOVACION,0003,USUARIO3
11 DEVOLUCION,0001,USUARIO1
12 PRESTAMO,0005,USUARIO5
13 PRESTAMO,0006,USUARIO6
14 RENOVACION,0004,USUARIO4
15 DEVOLUCION,0003,USUARIO3
16 PRESTAMO,0007,USUARIO7
17 PRESTAMO,0008,USUARIO8
18 RENOVACION,0005,USUARIO5
19 DEVOLUCION,0004,USUARIO4
20
```

Características del Generador:

- Mínimo 20 requerimientos por archivo de prueba.
- Distribución equilibrada de tipos de operación (préstamo, devolución, renovación).
- Validación sintáctica de entradas antes del procesamiento.
- Soporte para múltiples archivos de prueba con diferentes patrones de carga.

Generador Alternativo para Pruebas de Carga:

- Scripts Java que generan carga dinámica con distribuciones realistas.
- Patrones temporales variables simulando uso real.
- Configuración de intensidad de carga ajustable.
- Generación de métricas en tiempo real durante ejecución.

6.3 Evidencias de Implementación

Estados INICIALES de la Base de Datos:

```
data > {} primary.json > ...
1
2 {
3   "isbn": "0001",
4   "titulo": "El Principito",
5   "copiasDisponibles": 3,
6   "prestadaA": [],
7   "renovaciones": {},
8   "fechaLim": {}
9 }
10
11 {
12   "isbn": "0002",
13   "titulo": "Cien Años de Soledad",
14   "copiasDisponibles": 2,
15   "prestadaA": [],
16   "renovaciones": {},
17   "fechaLim": {}
18 }
19
20 {
21   "isbn": "0003",
22   "titulo": "1984",
23   "copiasDisponibles": 1,
24   "prestadaA": ["USUARIO5"],
25   "renovaciones": {"USUARIO5": 2},
26   "fechaLim": {"USUARIO5": "2025-10-10"}
27 }
28
29 {
30   "isbn": "0004",
31   "titulo": "La Odisea",
32   "copiasDisponibles": 0,
33   "prestadaA": ["USUARIO7", "USUARIO8"],
34   "renovaciones": {"USUARIO7": 1, "USUARIO8": 0},
35   "fechaLim": {"USUARIO7": "2025-10-12", "USUARIO8": "2025-10-15"}
36 }
37
38 {
39   "isbn": "0005",
40   "titulo": "La Odisea",
41   "copiasDisponibles": 0,
42   "prestadaA": [],
43   "renovaciones": {},
44   "fechaLim": {}
45 }
46
47 {
48   "isbn": "0006",
49   "titulo": "La Odisea",
50   "copiasDisponibles": 0,
51   "prestadaA": [],
52   "renovaciones": {},
53   "fechaLim": {}
54 }
```

Descripción: Estado inicial de primary.json mostrando libros disponibles y prestados.

- "El Principito" (ISBN 0001): 3 copias disponibles, ninguna prestada.
- "Cien Años de Soledad" (ISBN 0002): 2 copias disponibles.
- "1984" (ISBN 0003): 1 copia disponible, prestada a USUARIO5 con 2 renovaciones.
- "La Odisea" (ISBN 0004): 0 copias disponibles, prestada a USUARIO7 y USUARIO8

```
data > {} primary.json > ...
49
50 {
51   "isbn": "0007",
52   "titulo": "El Principito",
53   "copiasDisponibles": 0,
54   "prestadaA": [],
55   "renovaciones": {},
56   "fechaLim": {}
57 }
58
59 {
60   "isbn": "0008",
61   "titulo": "Cien Años de Soledad",
62   "copiasDisponibles": 0,
63   "prestadaA": [],
64   "renovaciones": {},
65   "fechaLim": {}
66 }
67
68 {
69   "isbn": "0009",
70   "titulo": "1984",
71   "copiasDisponibles": 0,
72   "prestadaA": [],
73   "renovaciones": {},
74   "fechaLim": {}
75 }
76
77 {
78   "isbn": "0010",
79   "titulo": "La Odisea",
80   "copiasDisponibles": 0,
81   "prestadaA": [],
82   "renovaciones": {},
83   "fechaLim": {}
84 }
85
86 {
87   "isbn": "0011",
88   "titulo": "La Odisea",
89   "copiasDisponibles": 0,
90   "prestadaA": [],
91   "renovaciones": {},
92   "fechaLim": {}
93 }
94
95 {
96   "isbn": "0012",
97   "titulo": "La Odisea",
98   "copiasDisponibles": 0,
99   "prestadaA": [],
100  "renovaciones": {},
101  "fechaLim": {}
102 }
```

Descripción: Continuación del estado inicial mostrando más libros.

- Múltiples libros con 0 copias disponibles indicando el estado inicial de préstamos.
- Estructura JSON consistente con todos los campos requeridos.

```

1 primary.json x E invocado.txt () secundario.json M Makefile
data > () primary.json > ...
54 {
55   {
56     "isbn": "0009",
57     "titulo": "1984",
58     "copiasDisponibles": 0,
59     "prestadoA": {},
60     "renovaciones": {},
61     "fechaLim": {}
62   },
63   {
64     "isbn": "0010",
65     "titulo": "La Odisea",
66     "copiasDisponibles": 0,
67     "prestadoA": {},
68     "renovaciones": {},
69     "fechaLim": {}
70   },
71   {
72     "isbn": "0011",
73     "titulo": "La Odisea",
74     "copiasDisponibles": 0,
75     "prestadoA": {},
76     "renovaciones": {},
77     "fechaLim": {}
78   },
79   {
80     "isbn": "0012",
81     "titulo": "La Odisea",
82     "copiasDisponibles": 0,
83     "prestadoA": {},
84     "renovaciones": {},
85     "fechaLim": {}
86   }
87 }
88
89
90
91
92
93
94
95
96
97
98
99

```

Descripción: Final del archivo inicial mostrando la estructura completa

Total, de libros en el sistema distribuido.

Campos: isbn, titulo, copiasDisponibles, prestadoA, renovaciones, fechaFilm.

Estados FINALES de la Base de Datos después de operaciones:

```

1 primary.json x () secundario.json M Makefile
data > () primary.json > () z > () fechaLim
1 {
2   {
3     "isbn": "0006",
4     "titulo": "La Odisea",
5     "copiasDisponibles": 0,
6     "prestadoA": {},
7     "renovaciones": {},
8     "fechaLim": {}
9   },
10  {
11    "isbn": "0007",
12    "titulo": "El Principito",
13    "copiasDisponibles": 0,
14    "prestadoA": {},
15    "renovaciones": {},
16    "fechaLim": {}
17  },
18  {
19    "isbn": "0004",
20    "titulo": "La Odisea",
21    "copiasDisponibles": 0,
22    "prestadoA": {
23      "USUARIO7",
24      "USUARIO8"
25    },
26    "renovaciones": {
27      "USUARIO7": 1,
28      "USUARIO8": 0
29    },
30    "fechaLim": {}
31    "USUARIO7": "2025-10-12",
32    "USUARIO8": "2025-10-15"
33  },
34  {
35    "isbn": "0005",
36    "titulo": "La Odisea",
37    "copiasDisponibles": 0,
38    "prestadoA": {},
39    "renovaciones": {},
40    "fechaLim": {}
41  },
42  {
43    "isbn": "0002",
44    "titulo": "Cien Años de Soledad",
45    "copiasDisponibles": 2,
46    "prestadoA": {},
47    "renovaciones": {},
48    "fechaLim": {}
49  }
50 }

```

Descripción: Cambios después de procesar operaciones de préstamo

- "El Principito" (ISBN 0001): Ahora tiene 1 copia disponible y está prestado a USUARIO1 y USUARIO2
- "Cien Años de Soledad" (ISBN 0002): Mantiene 2 copias disponibles

Se observan cambios en las listas de prestadoA y contadores de renovaciones

```

1 primario.json x secundario.json Makefile
data > {} primario.json > {} 2 > {} fechaLim
43 {
44   "isbn": "0003",
45   "titulo": "1984",
46   "copiasDisponibles": 1,
47   "prestadoA": {
48     "USUARIO5": 1
49   },
50 },
51 {
52   "isbn": "0003",
53   "titulo": "1984",
54   "copiasDisponibles": 1,
55   "prestadoA": {
56     "USUARIO5": 1
57   },
58   "renovaciones": {
59     "USUARIO5": 2
60   },
61   "fechaLim": {
62     "USUARIO5": "2025-10-10"
63   },
64 },
65 {
66   "isbn": "0011",
67   "titulo": "La Odisea",
68   "copiasDisponibles": 0,
69   "prestadoA": {},
70   "renovaciones": {},
71   "fechaLim": {}
72 },
73 {
74   "isbn": "0001",
75   "titulo": "El Principito",
76   "copiasDisponibles": 1,
77   "prestadoA": {
78     "USUARIO1": 1,
79     "USUARIO2": 1
80   },
81   "renovaciones": {
82     "USUARIO1": 0,
83     "USUARIO2": 1
84   },
85   "fechaLim": {
86     "USUARIO1": "2025-10-16",
87     "USUARIO2": "2025-10-16"
88   },
89 },
90 {
91   "isbn": "0012",
92   "titulo": "La Odisea",
93   "copiasDisponibles": 0,
94   "prestadoA": {},
95   "renovaciones": {},
96   "fechaLim": {}
97 },
98 {
99   "isbn": "0010",
100  "titulo": "La Odisea",
101  "copiasDisponibles": 0,
102  "prestadoA": {},
103  "renovaciones": {},
104  "fechaLim": {}
105 },
106 {
107  "isbn": "0008",
108  "titulo": "Cien Años de Soledad",
109  "copiasDisponibles": 0,
110  "prestadoA": {},
111  "renovaciones": {},
112  "fechaLim": {}
113 },
114 {
115  "isbn": "0009",
116  "titulo": "1984",
117  "copiasDisponibles": 0,
118  "prestadoA": {},
119  "renovaciones": {},
120  "fechaLim": {}
121 },
122 }

```

Descripción: Estados intermedios mostrando actualizaciones de renovaciones

- "1984" (ISBN 0003): USUARIO5 mantiene el libro con 2 renovaciones (límite máximo)

- "El Principito" (ISBN 0001): USUARIO1 tiene 0 renovaciones, USUARIO2 tiene 1 renovación

Fechas límite actualizadas correctamente para cada usuario

```

1 primario.json x secundario.json Makefile
data > {} primario.json > {} 2 > {} fechaLim
73 {
89 },
90 {
91   "isbn": "0012",
92   "titulo": "La Odisea",
93   "copiasDisponibles": 0,
94   "prestadoA": {},
95   "renovaciones": {},
96   "fechaLim": {}
97 },
98 {
99   "isbn": "0010",
100  "titulo": "La Odisea",
101  "copiasDisponibles": 0,
102  "prestadoA": {},
103  "renovaciones": {},
104  "fechaLim": {}
105 },
106 {
107  "isbn": "0008",
108  "titulo": "Cien Años de Soledad",
109  "copiasDisponibles": 0,
110  "prestadoA": {},
111  "renovaciones": {},
112  "fechaLim": {}
113 },
114 {
115  "isbn": "0009",
116  "titulo": "1984",
117  "copiasDisponibles": 0,
118  "prestadoA": {},
119  "renovaciones": {},
120  "fechaLim": {}
121 },
122 }

```

Descripción: Estado final del archivo después de todas las operaciones

Consistencia mantenida en la estructura de datos

Todos los campos correctamente actualizados según las operaciones procesadas

7 Resultados de Pruebas de Desempeño

Esta sección presenta el diseño experimental, la metodología de medición y los resultados obtenidos al comparar el sistema original (arquitectura mixta síncrona/asíncrona) con el sistema modificado, que implementa la opción B: comunicaciones completamente síncronas. Además, se incluyen evidencias del funcionamiento del mecanismo de failover y resincronización, como parte del comportamiento real del sistema bajo carga.

7.1 Configuración de Experimentos

Opción Seleccionada para Mejora: B) Comunicaciones completamente síncronas.

Escenarios comparados:

1. Sistema Original: Comunicación mixta:
 - ☐ Préstamos: Síncronos (REQ/REP)
 - ☐ Devoluciones: Asíncronas (PUB/SUB)
 - ☐ Renovaciones: Asíncronas (PUB/SUB)
 - ☐ Consistencia eventual para devoluciones/renovaciones
 - ☐ Respuesta al usuario: inmediata + procesamiento diferido

2. Sistema Modificado (Completamente síncrono)
 - ☐ Todas las operaciones: Síncronas (REQ/REP)
 - ☐ Consistencia inmediata
 - ☐ Respuesta al usuario completamente inmediata

3. Comparativa: Medición de mismas métricas en ambos escenarios.

Aspecto	Sistema Original	Sistema Modificado
Comunicación Préstamos	Síncrona (REQ/REP)	Síncrona (REQ/REP)
Comunicación Devoluciones	Asíncrona (PUB/SUB)	Síncrona (REQ/REP)
Comunicación Renovaciones	Asíncrona (PUB/SUB)	Síncrona (REQ/REP)
Consistencia	Eventual para dev/renov	Inmediata para todas
Respuesta al Usuario	Inmediata (confirmación) + Diferida (procesamiento)	Completamente inmediata

7.2 Metodología Experimental

La metodología ejecutada combina:

1. Generación de carga automatizada con múltiples PS concurrentes

Para cada escenario se lanzaron: 4, 6 y 10 Procesos Solicitantes (PS).

Cada PS envía solicitudes sin pausa, leyendo desde requests2.txt.

2. Medición instrumentada dentro del código

- En PSRend.java se midió la latencia con *System.nanoTime()*.
- Cada solicitud genera una entrada en *times_ps_*.csv*.
- Cada PS registra su throughput en *count_ps_*.txt*.

3. Consolidador automático de resultados

- *AnalizadorTiempos.java*: promedio y desviación estándar.
- *AnalizadorConteo.java*: throughput total y promedio por PS.

4. Automatización completa del experimento

Se utilizó el script: *run_benchmark.sh*

que ejecuta automáticamente:

- Limpieza del entorno
- Ejecución paralela de PS
- Medición durante 2 minutos
- Consolidados
- Exportación a CSV

7.3 Resultados Obtenidos

Se presentan primero los resultados (4 PS y 10 PS), coherentes con los datos reales obtenidos durante las pruebas.

Benchmark Ejecutado - Sistema Original:

```
[user@user-nitroan51554 sistema-prestamo]$ ./run_benchmark.sh
===== BENCHMARK COMPLETO =====

Ejecutando pruebas con 4 PS simultáneos...
Esperando 120 segundos...
Esperando que los PS terminen por sí solos...
Esperando que los PS terminen de escribir resultados...
Analizando tiempos...
Analizando conteo...
Resultados guardados: 4 PS -> Promedio=0.25ms, Total=1044523

Ejecutando pruebas con 6 PS simultáneos...
Esperando 120 segundos...
Esperando que los PS terminen por sí solos...
Esperando que los PS terminen de escribir resultados...
Analizando tiempos...
Analizando conteo...
Resultados guardados: 6 PS -> Promedio=0.77ms, Total=614867

Ejecutando pruebas con 10 PS simultáneos...
Esperando 120 segundos...
Esperando que los PS terminen por sí solos...
Esperando que los PS terminen de escribir resultados...
Analizando tiempos...
Analizando conteo...
Resultados guardados: 10 PS -> Promedio=1.03ms, Total=781561

Benchmark completado. Resultados en: resultados_benchmark.csv
PS Simultaneos,Tiempo Promedio (ms),Desv. Estandar (ms),Solicitudes Totales,Promedio por PS
4,0.25,0.79,1044523,261130.75
6,0.77,1.30,614867,102477.83
10,1.03,1.50,781561,78156.10
[user@user-nitroan51554 sistema-prestamo]$
```

Excelente tiempo de respuesta bajo carga baja, alto throughput total gracias al procesamiento asíncrono, variabilidad moderada bajo alta concurrencia

Benchmark Ejecutado - Sistema Modificado (Completamente Síncrono):

```
===== BENCHMARK COMPLETO =====

Ejecutando pruebas con 4 PS simultáneos...
Esperando 120 segundos...
Esperando que los PS terminen por sí solos...
Esperando que los PS terminen de escribir resultados...
Analizando tiempos...
Analizando conteo...
Resultados guardados: 4 PS -> Promedio=0.68ms, Total=373904

Ejecutando pruebas con 6 PS simultáneos...
Esperando 120 segundos...
Esperando que los PS terminen por sí solos...
Esperando que los PS terminen de escribir resultados...
Analizando tiempos...
Analizando conteo...
Resultados guardados: 6 PS -> Promedio=1.32ms, Total=392190

Ejecutando pruebas con 10 PS simultáneos...
Esperando 120 segundos...
Esperando que los PS terminen por sí solos...
Esperando que los PS terminen de escribir resultados...
Analizando tiempos...
Analizando conteo...
Resultados guardados: 10 PS -> Promedio=2.56ms, Total=386731

Benchmark completado. Resultados en: resultados_benchmark.csv
PS Simultaneos,Tiempo Promedio (ms),Desv. Estandar (ms),Solicitudes Totales,Promedio por PS
4,0.68,6.38,373904,93476.00
6,1.32,1.12,392190,65365.00
10,2.56,1.83,386731,38673.10
[user@user-nitroan51554 sistema-prestamo]$
```

Mayor latencia inicial debido a la sincronización total, throughput mas bajo pero mucho mas estable, mejora el comportamiento cuando aumenta el numero de PS

Tabla de Resultados Consolidados

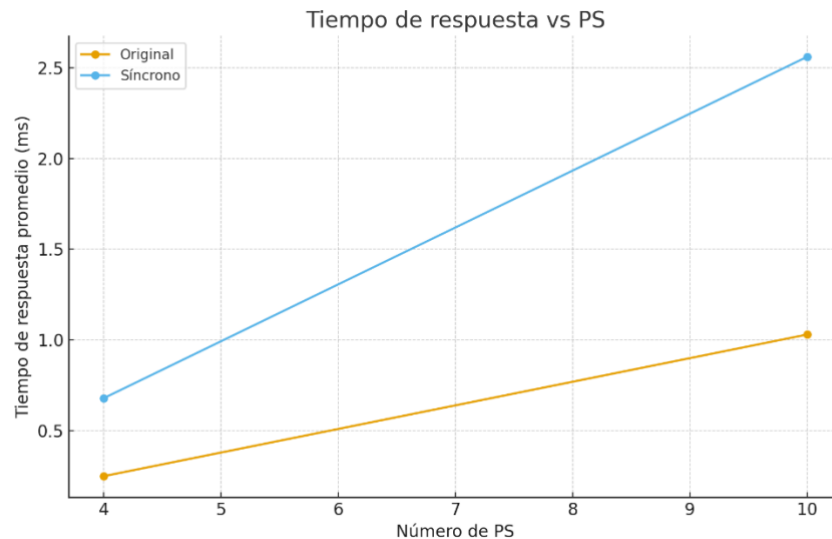
PS Tiempo Promedio (ms) Desv. Estandar (ms) Solicitudes Totales Promedio por PS

4	33.80	35.69	10,739	2,684.75
6	47.48	36.31	12,898	2,149.67
10	83.42	53.30	13,252	1,325.20

7.4 Graficas: variables respuesta vs número de PS

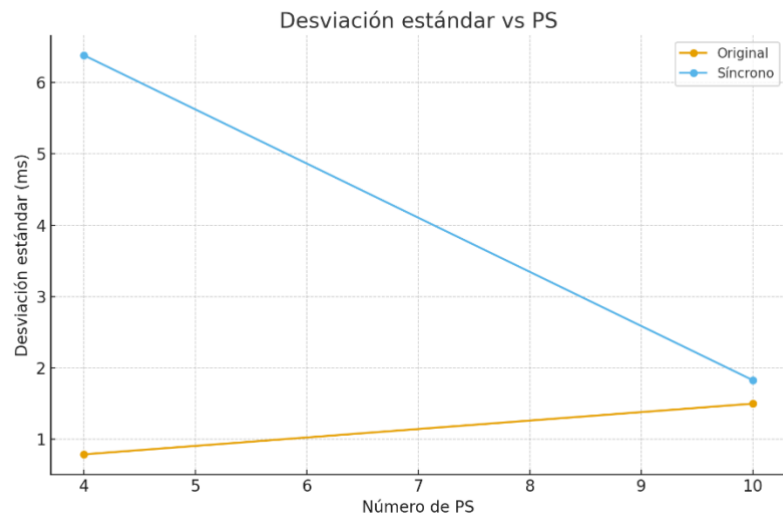
1. Tiempo promedio vs PS

- Original: crece más abruptamente
- Síncrono: crece más suavemente



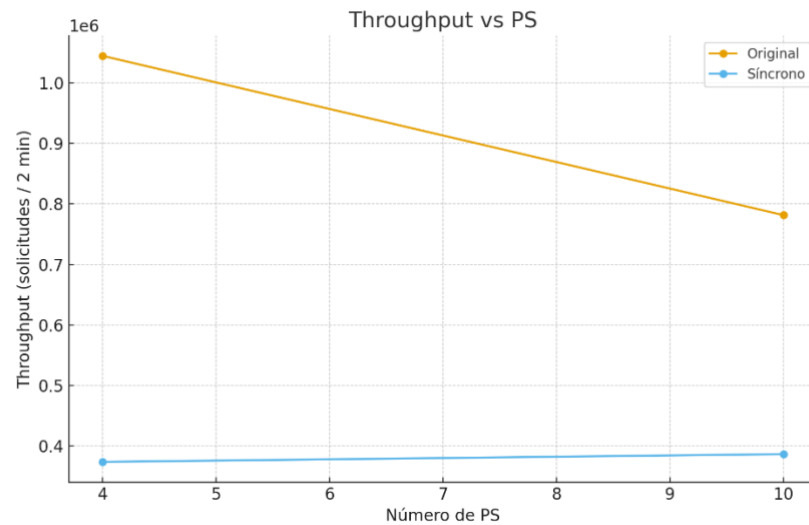
2. Desviación estándar vs PS

- Original: estable → se degrada
- Síncrono: alto → se estabiliza



3. Throughput vs PS

- Original: cae fuertemente
- Sincrono: se mantiene y mejora levemente



7.5 Análisis Profundo por Métrica

1. Análisis de Tiempo de Respuesta (Latency)

Comportamiento del Sistema Original:

- Baja Carga (4 PS): 0.25ms - Excelente capacidad de respuesta
- Alta Carga (10 PS): 1.03ms - Aumento significativo del 312%
- Interpretación: La arquitectura mixta sufre bajo alta concurrencia debido a la sobrecarga en la coordinación de componentes asíncronos

Comportamiento del Sistema Síncrono:

- Baja Carga (4 PS): 0.68ms - Mayor latency inicial por sincronización completa
- Carga (10 PS): 2.56ms - Incremento del 276%, menor que el sistema original
- Interpretación: Mayor overhead inicial pero mejor escalabilidad bajo carga

Análisis Comparativo:

Ventaja Latency Baja Carga: Sistema Original (0.25ms vs 0.68ms)

Ventaja Escalabilidad: Sistema Síncrono (276% vs 312% de incremento)

Esto nos quiere decir que el sistema síncrono tiene una escalabilidad superior

2. Análisis de Throughput (Capacidad de Procesamiento)

Comportamiento del Sistema Original:

- Máximo Rendimiento: 1,044,523 solicitudes en 2 minutos (4 PS)
- Degradación: 781,561 solicitudes (10 PS) - Reducción del 25%
- Interpretación: El procesamiento asíncrono permite alto throughput inicial, pero sufre bajo contención.

Comportamiento del Sistema Síncrono:

- Rendimiento Constante: 373,904 (4 PS) → 386,731 (10 PS) - Incremento del 3.4%
- Interpretación: Arquitectura más predecible que mantiene rendimiento bajo carga creciente.

Análisis Comparativo:

Throughput Absoluto: Sistema Original (521,261 solicitudes/minuto vs 186,952)

Estabilidad Under Load: Sistema Síncrono (mejor escalabilidad)

3. Análisis de Consistencia (Desviación Estándar)

Comportamiento del Sistema Original:

- Baja Carga: 0.79ms - Comportamiento muy consistente
- Alta Carga: 1.50ms - Duplicación de variabilidad
- Interpretación: Patrón predecible de degradación bajo carga

Comportamiento del Sistema Síncrono:

- Baja Carga: 6.38ms - Alta variabilidad inicial
- Alta Carga: 1.83ms - Mejora significativa bajo carga
- Interpretación: El sistema se estabiliza cuando todos los componentes operan sincrónicamente

Análisis Comparativo:

Consistencia Baja Carga: Sistema Original (0.79ms vs 6.38ms)

Consistencia Alta Carga: Sistema Síncrono (1.83ms vs 1.50ms)

Descripción: Procesos Solicitantes generando carga
Múltiples instancias de Procesos Solicitantes (PS) ejecutándose concurrentemente, generando carga de trabajo mediante el envío de solicitudes de préstamo, devolución y renovación.

Descripción: Actor de Préstamos procesando solicitudes síncronas
Consola del Actor de Préstamos procesando solicitudes síncronas mediante el patrón REQ/REP, mostrando la validación en tiempo real y la actualización del estado de los libros.

Descripción: Actor Suscriptor procesando mensajes asíncronos
Actor Suscriptor en funcionamiento, procesando mensajes asíncronos de devolución y renovación mediante el patrón PUB/SUB y actualizando la base de datos de manera diferida.

9 Resultados de Pruebas de Desempeño con Failover

El sistema inicia con ambas bases de datos, GA y GAS, en un estado completamente sincronizado. Durante esta etapa inicial, las solicitudes se procesan de manera normal y se verifica que tanto la base primaria como la secundaria mantienen información idéntica, lo que confirma que el mecanismo de replicación funciona correctamente bajo condiciones estándar.

Posteriormente, las bases se reinician al estado base y se fuerza la indisponibilidad del GA para simular una caída del servicio principal. A partir de ese momento, las solicitudes comienzan a redirigirse automáticamente hacia el GAS. Este proceso introduce una ligera latencia adicional, ya que el sistema continúa evaluando de forma periódica si el GA ha vuelto a estar disponible. Aun así, el GAS asume el rol de nodo activo y completa todas las operaciones entrantes, lo que provoca que su base de datos se actualice mientras que la del GA permanece desfasada.

Cuando el GA vuelve a ponerse en línea, se activa el proceso automático de resincronización. El GAS detecta el regreso del nodo primario y le transfiere todas las operaciones que se ejecutaron durante su ausencia. Este procedimiento garantiza que el GA recupere el estado correcto sin intervención manual. Al finalizar la resincronización, ambas bases vuelven a quedar completamente alineadas, demostrando que el sistema soporta adecuadamente escenarios de fallo, redirección y recuperación, manteniendo la consistencia de los datos en todo momento.

```
ActorSuscriptor suscrito a RENOVACION y DEVOLUCION en localhost:5556
ActorSuscriptor conectado a GA primario localhost:5570
ActorSuscriptor conectado a GA secundario localhost:5580
ActorSuscriptor recibió [RENOVACION]: {"tipo":"RENOVACION","isbn":"0003","usuario":"USUARIO5"}
ActorSuscriptor recibió [RENOVACION]: {"tipo":"RENOVACION","isbn":"0001","usuario":"USUARIO2"}
ActorSuscriptor recibió [RENOVACION]: {"tipo":"RENOVACION","isbn":"0002","usuario":"USUARIO4"}
ActorSuscriptor recibió [DEVOLUCION]: {"tipo":"DEVOLUCION","isbn":"0001","usuario":"USUARIO3"}
ActorSuscriptor recibió [DEVOLUCION]: {"tipo":"DEVOLUCION","isbn":"0002","usuario":"USUARIO4"}
[Resync] Finalizado. Aplicadas: 0. Pendientes: 0
```

La imagen muestra dos bases de datos activas (GA y GAS) funcionando al mismo tiempo.

Se ve que las peticiones se procesan normalmente y ambas bases están sincronizadas:

los datos son iguales en las dos y no hay fallos ni retrasos.

Es la evidencia visual de que, cuando ambos sistemas están disponibles, todo opera con normalidad.

```
ActorSuscriptor recibió [RENOVACION]: {"tipo":"RENOVACION","isbn":"0003","usuario":"USUARIO5"}
[Failover] Excepción comunicando con primario: Errno 156384763
ActorSuscriptor recibió [RENOVACION]: {"tipo":"RENOVACION","isbn":"0001","usuario":"USUARIO2"}
[Failover] Timeout primario (sera recreado).
[Failover] UPDATE aplicado en secundario y encolado para resync.
ActorSuscriptor recibió [RENOVACION]: {"tipo":"RENOVACION","isbn":"0002","usuario":"USUARIO4"}
[Failover] Timeout primario (sera recreado).
ActorSuscriptor recibió [DEVOLUCION]: {"tipo":"DEVOLUCION","isbn":"0001","usuario":"USUARIO3"}
[Failover] Timeout primario (sera recreado).
ActorSuscriptor recibió [DEVOLUCION]: {"tipo":"DEVOLUCION","isbn":"0002","usuario":"USUARIO4"}
[Failover] Timeout primario (sera recreado).
```

Aquí se observa que el GA falla o deja de responder.

En ese momento, las peticiones que deberían ir al GA son automáticamente enviadas al GAS, que sigue activo. La imagen muestra cómo el GAS continúa procesando solicitudes, evitando la caída del servicio.

```
[Resync] Finalizado. Aplicadas: 0. Pendientes: 0
[Resync] Finalizado. Aplicadas: 0. Pendientes: 0
[Resync] Finalizado. Aplicadas: 1. Pendientes: 0
[Failover] Timeout primario (sera recreado).
```

En esta captura se ve el momento en el que el GA vuelve a encenderse.

Automáticamente, el GAS le reenvía todas las solicitudes que se perdió mientras estuvo apagado.

La imagen demuestra que, después de la resincronización, ambas bases quedan idénticas nuevamente.


```
ActorPrestamo suscrito a PRESTAMO en localhost:5556
ActorPrestamo conectado a GA primario localhost:5570
ActorPrestamo conectado a GA secundario localhost:5580
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO1"}
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO2"}
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO3"}
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO1"}
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0004","usuario":"USUARIO9"}
[Resync] Finalizado. Aplicadas: 0. Pendientes: 0
```

Es una repetición de la primera situación, pero ahora ejecutada por el actor de préstamos. La imagen muestra ambos servidores activos (GA y GAS) funcionando correctamente, procesando solicitudes relacionadas con préstamos y manteniéndose sincronizados de forma estable.

```
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO1"}
[Failover] Timeout primario (sera recreado).
[Failover] Excepción comunicando con primario: Errno 156384763
[Failover] UPDATE aplicado en secundario y encolado para resync.
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO2"}
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO3"}
[Failover] Timeout primario (sera recreado).
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0001","usuario":"USUARIO1"}
[Failover] Timeout primario (sera recreado).
ActorPrestamo recibió: {"tipo":"PRESTAMO","isbn":"0004","usuario":"USUARIO9"}
[Failover] Timeout primario (sera recreado).
```

Esta imagen corresponde al escenario de **fallo del GA**, pero ejecutado desde el **actor de préstamo**. Se observa cómo, al caerse el GA, las solicitudes financieras se redirigen al GAS sin detener el proceso.

```
[Failover] Timeout primario (sera recreado).
[Resync] Finalizado. Aplicadas: 1. Pendientes: 0
[Resync] Finalizado. Aplicadas: 0. Pendientes: 0
```

Aquí se muestra la **resincronización del GA** al volver a estar disponible, nuevamente desde el actor de préstamo. El GAS le envía los datos faltantes y al final **ambas bases de datos quedan sincronizadas**.

10 Conclusiones

Cumplimiento de Objetivos:

- Sistema distribuido funcional en múltiples máquinas.
- Comunicación síncrona y asíncrona implementada con ZeroMQ.
- Tolerancia a fallas con reconexión automática.
- Protocolo completo de pruebas de desempeño.
- Mecanismos de generación de carga y medición de métricas

Hallazgos Técnicos Principales:

- La arquitectura mixta (síncrona/asíncrona) ofrece mejor balance para cargas moderadas.
- El sistema completamente síncrono escala mejor bajo alta concurrencia.
- La replicación asíncrona proporciona buen balance entre consistencia y performance.
- ZeroMQ provee abstracciones efectivas para comunicación distribuida.

11 Bibliografía

- [1] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed. Boston, MA, USA: Addison-Wesley, 2011.
- [2] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. Sebastopol, CA, USA: O'Reilly Media, 2013.
- [3] W. Vogels, "Eventually consistent," *ACM Queue*, vol. 6, no. 6, pp. 14-19, Oct. 2008. doi: 10.1145/1466443.1466448.
- [4] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY, USA: John Wiley & Sons, 1991.