PRIMERA ENTREGA PROYECTO


# Sistemas Distribuidos de Préstamo de Libros

Arley Bernal Muñetón
Kevin Garay Gaitán
Juan Esteban Bello

Introducción a Sistemas Distribuidos
5 de octubre 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 asincrónica 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:**

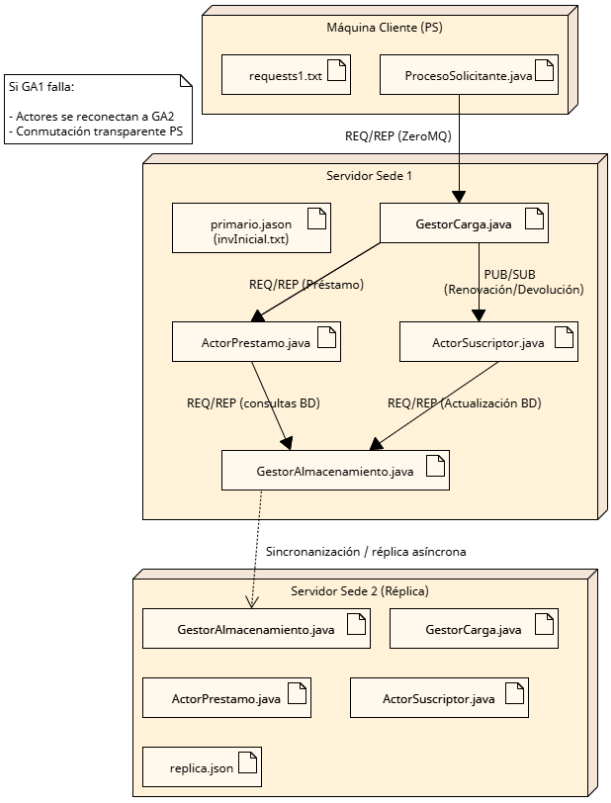
- Validación de reglas de negocio (máximo 2 renovaciones por libro)
- Consistencia en actualizaciones de inventario
- Prevención de condiciones de carrera en operaciones concurrentes

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



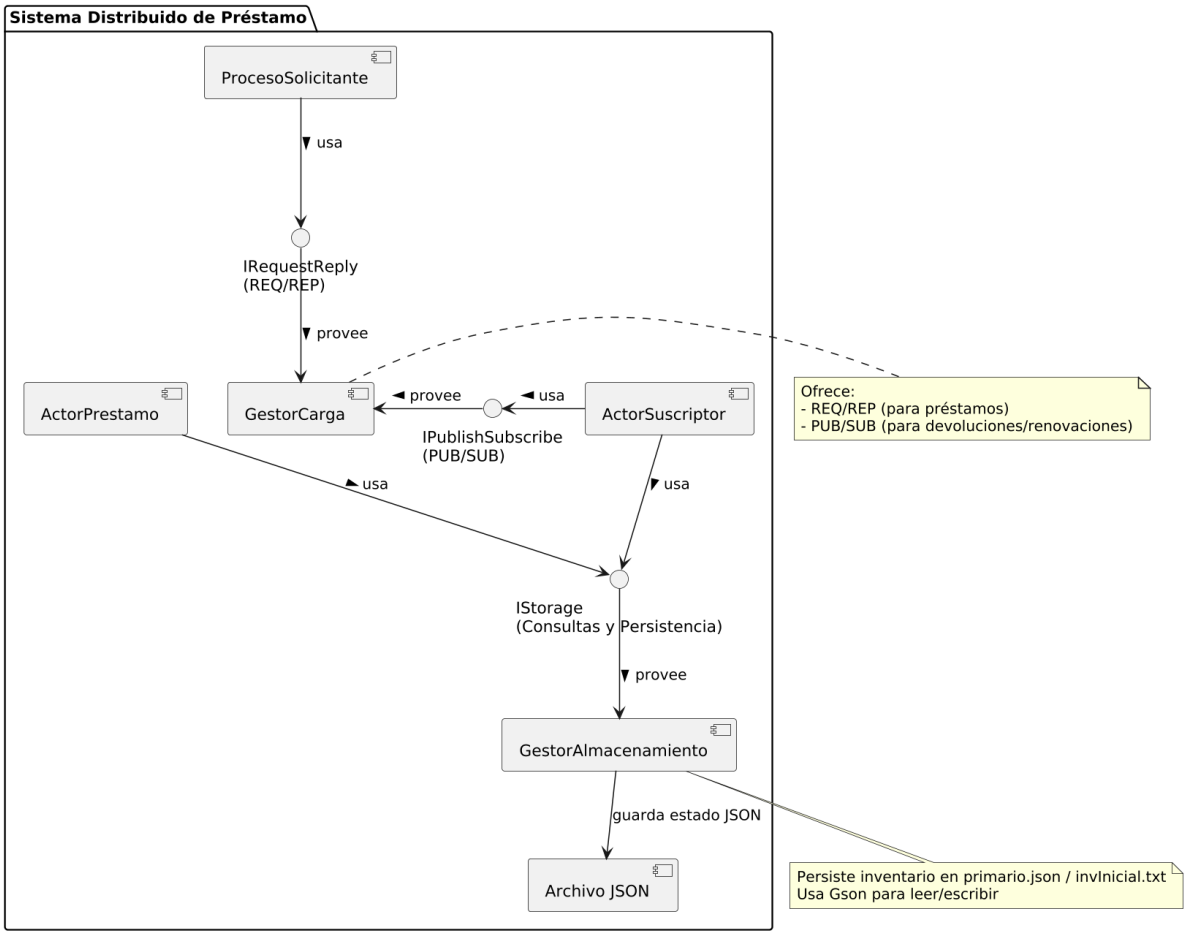

Descripción del despliegue:

- **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



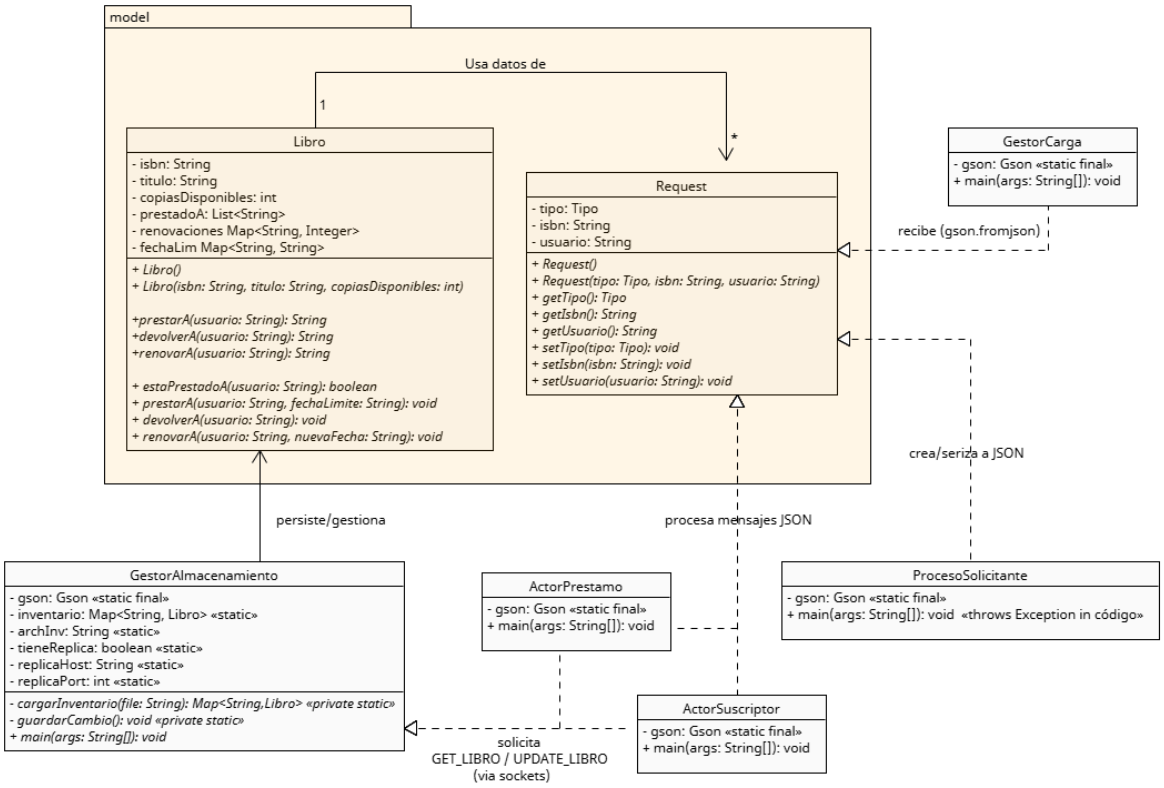

Descripción de componentes:

- **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



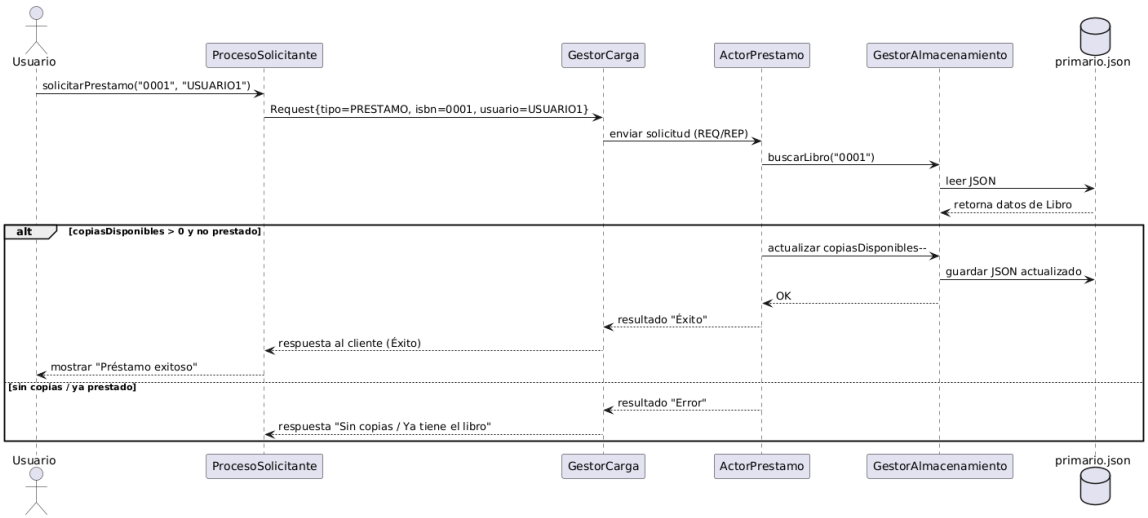

Descripción del modelo de datos:

- 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 → primario.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:<br>- T. Respuesta promedio (y desv. standard) de las solicitudes de préstamo.<br>- Cantidad de solicitudes procesadas (respuesta dada a los PS) en un periodo de 2min. | Variables a medir:<br>- T. Respuesta promedio (y desv. standard) de las solicitudes de préstamo.<br>- 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:

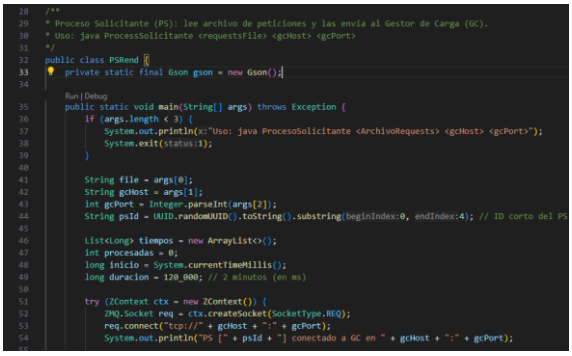
**Ventajas del Enfoque Implementado:**

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.

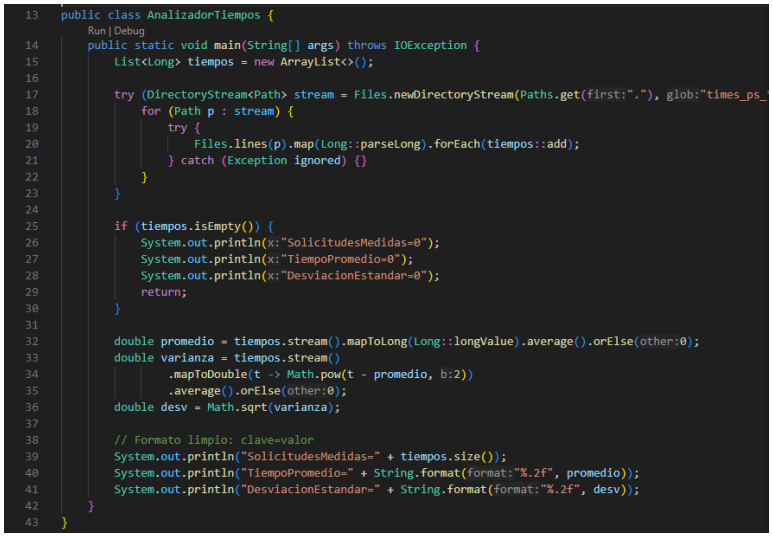
**Arquitectura de Medición Implementada**

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.

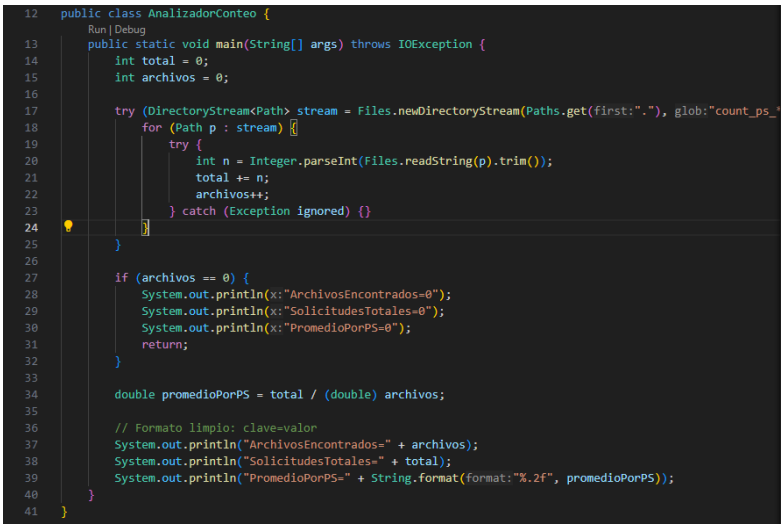


2. AnalizadorTiempos.java - Procesador de Métricas de Tiempo:
    - Agregación: Consolida todos los archivos times_ps_*.csv
    - Cálculo de Promedio: tiempos.stream().mapToLong(Long::longValue).average()
    - Desviación Estándar: Cálculo matemático de varianza y raíz cuadrada
    - Formato de Salida: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



```java
public class AnalizadorTiempos {
    Run | Debug
    public static void main(String[] args) throws IOException {
        List<Long> tiempos = new ArrayList<>();

        try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get(first:"."), glob:"times_ps_
            for (Path p : stream) {
                try {
                    Files.lines(p).map(Long::parseLong).forEach(tiempos::add);
                } catch (Exception ignored) {}
            }
        }

        if (tiempos.isEmpty()) {
            System.out.println(x:"SolicitudesMedidas=0");
            System.out.println(x:"TiempoPromedio=0");
            System.out.println(x:"DesviacionEstandar=0");
            return;
        }

        double promedio = tiempos.stream().mapToLong(Long::longValue).average().orElse(other:0);
        double varianza = tiempos.stream()
                .mapToDouble(t -> Math.pow(t - promedio, b:2))
                .average().orElse(other:0);
        double desv = Math.sqrt(varianza);

        // Formato limpio: clave=valor
        System.out.println("SolicitudesMedidas=" + tiempos.size());
        System.out.println("TiempoPromedio=" + String.format(format:"%.2f", promedio));
        System.out.println("DesviacionEstandar=" + String.format(format:"%.2f", desv));
    }
}
```

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



```java
public class AnalizadorConteo {
    Run | Debug
    public static void main(String[] args) throws IOException {
        int total = 0;
        int archivos = 0;

        try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get(first:"."), glob:"count_ps_
            for (Path p : stream) {
                try {
                    int n = Integer.parseInt(Files.readString(p).trim());
                    total += n;
                    archivos++;
                } catch (Exception ignored) {}
            }
        }

        if (archivos == 0) {
            System.out.println(x:"ArchivosEncontrados=0");
            System.out.println(x:"SolicitudesTotales=0");
            System.out.println(x:"PromedioPorPS=0");
            return;
        }

        double promedioPorPS = total / (double) archivos;

        // Formato limpio: clave=valor
        System.out.println("ArchivosEncontrados=" + archivos);
        System.out.println("SolicitudesTotales=" + total);
        System.out.println("PromedioPorPS=" + String.format(format:"%.2f", promedioPorPS));
    }
}
```

**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(other:0);
33      double varianza = tiempos.stream()
34              .mapToDouble(t -> Math.pow(t - promedio, b:2))
35              .average().orElse(other: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**

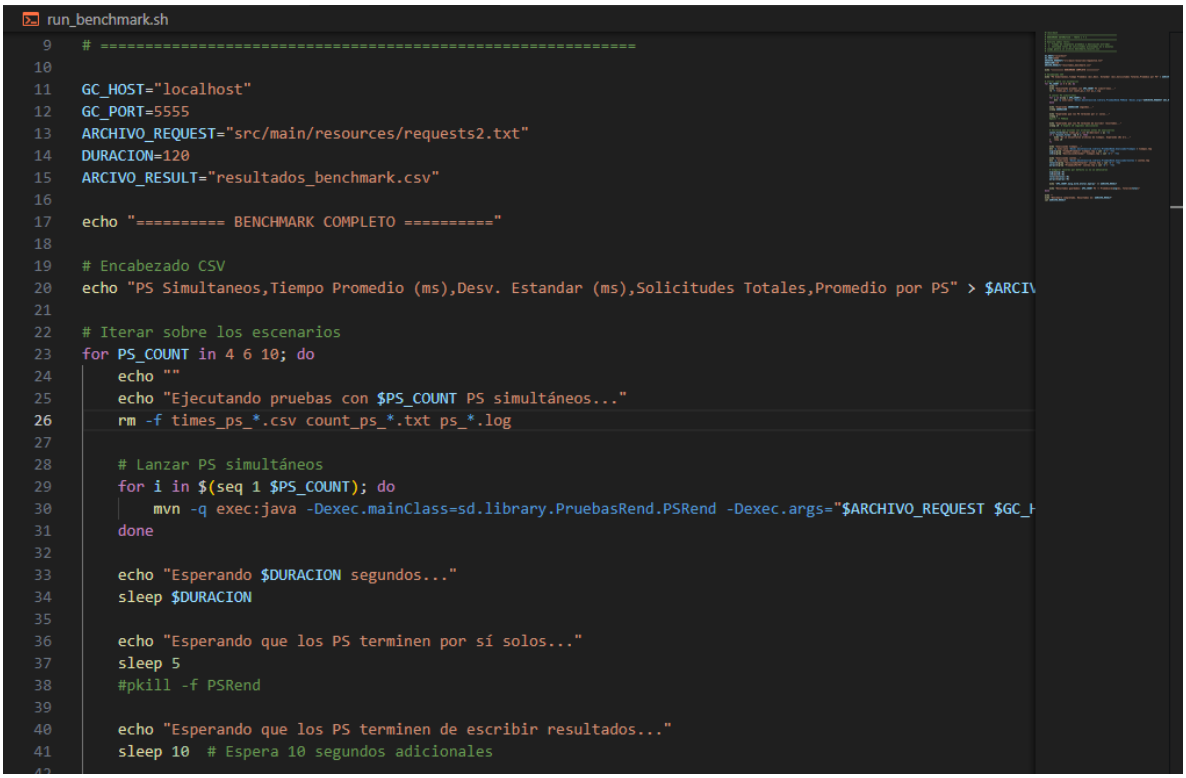
Script de Automatización: run_benchmark.sh

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.




**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):**

```
# Formato por línea: Tipo, ISBN, Usuario
# TIPO = PRESTAMO | DEVOLUCION | RENOVACION

#Éxito
PRESTAMO,0001,USUARIO1
#Éxito
PRESTAMO,0001,USUARIO2
#Éxito (quedan 0 copias)
PRESTAMO,0001,USUARIO3
#Error: ya tiene el libro
PRESTAMO,0001,USUARIO1
#Error: sin copias disponibles
PRESTAMO,0004,USUARIO9

#Error: máximo de renovaciones alcanzado
RENOVACION,0003,USUARIO5
#Éxito
RENOVACION,0001,USUARIO2
#Error: no tiene el libro
RENOVACION,0002,USUARIO4

#Éxito
DEVOLUCION,0001,USUARIO3
#Error: no tiene el libro
DEVOLUCION,0002,USUARIO4
```

```
# Archivo con peticiones de ejemplo (mínimo 20)
# Formato por línea: Tipo, ISBN, Usuario
# TIPO = PRESTAMO | DEVOLUCION | RENOVACION
PRESTAMO,0001,USUARIO1
PRESTAMO,0002,USUARIO2
PRESTAMO,0003,USUARIO3
PRESTAMO,0004,USUARIO4
RENOVACION,0001,USUARIO1
DEVOLUCION,0002,USUARIO2
RENOVACION,0003,USUARIO3
DEVOLUCION,0001,USUARIO1
PRESTAMO,0005,USUARIO5
PRESTAMO,0006,USUARIO6
RENOVACION,0004,USUARIO4
DEVOLUCION,0003,USUARIO3
PRESTAMO,0007,USUARIO7
PRESTAMO,0008,USUARIO8
RENOVACION,0005,USUARIO5
DEVOLUCION,0004,USUARIO4

```

**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**:


Descripción: Estado inicial de primario.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


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.


Descripción: Final del archivo inicial mostrando la estructura completa

Total, de libros en el sistema distribuido.

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

**Estados FINALES de la Base de Datos después de operaciones:**


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

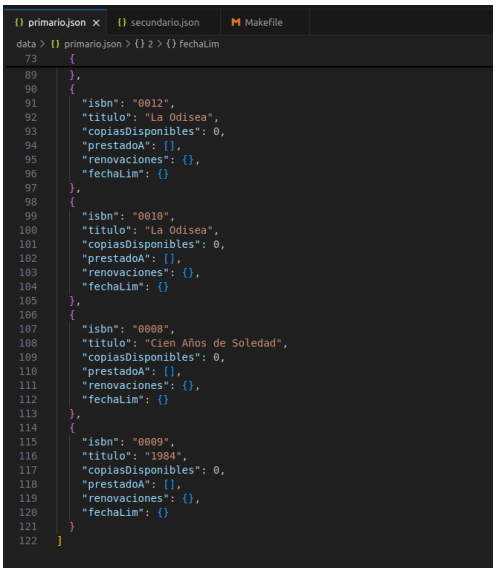
Se observan cambios en las listas de prestadoA y contadores de renovaciones


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




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

## 7.1 Configuración de Experimentos

Opción Seleccionada para Mejora: B) Comunicaciones completamente síncronas.

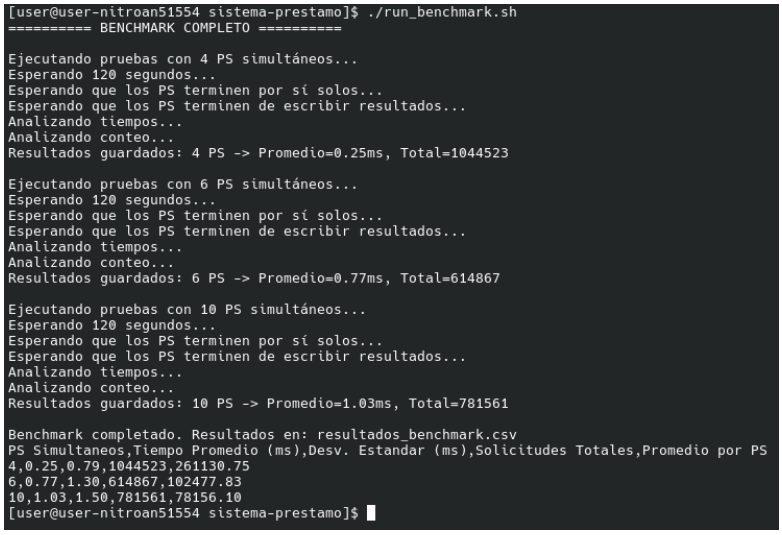
**Metodología Experimental:**

1. Sistema Original: Comunicación mixta (síncrona para préstamos, asíncrona para demás).
2. Sistema Modificado: Todas las comunicaciones implementadas como síncronas.
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 Resultados Obtenidos

**Benchmark Ejecutado - Sistema Original:**



**Benchmark Ejecutado - Sistema Modificado (Completamente Síncrono):**

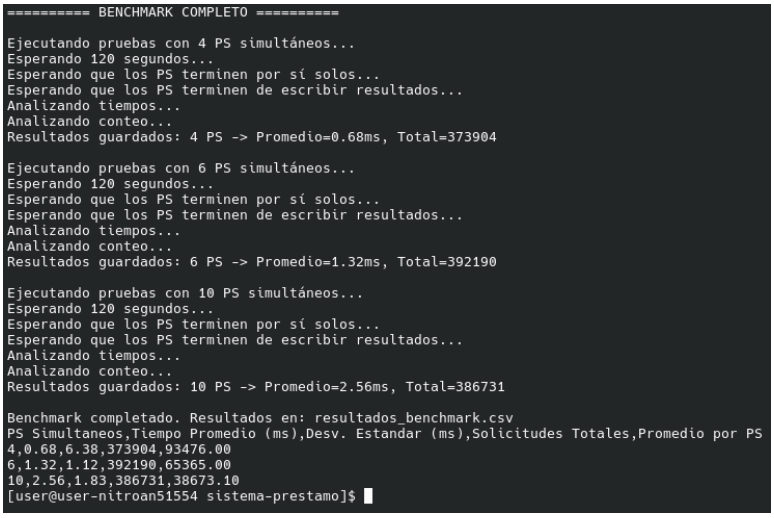



**Tabla de Resultados Consolidados**

| Métrica | Sistema Original (4 PS) | Sistema Original (10 PS) | Sistema Síncrono (4 PS) | Sistema Síncrono (10 PS) |
|---|---|---|---|---|
| **Tiempo Respuesta Promedio (ms)** | 0.25 | 1.03 | 0.68 | 2.56 |
| **Incremento Tiempo Respuesta** | - | 312% | - | 276% |
| **Throughput Total (solicitudes/2min)** | 1,044,523 | 781,561 | 373,904 | 386,731 |
| **Variación Throughput** | - | -25% | - | 3.40% |
| **Desviación Estándar (ms)** | 0.79 | 1.5 | 6.38 | 1.83 |
| **Comportamiento Consistencia** | Estable | Degradación progresiva | Alta variabilidad | Estabilización |

## 7.3 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 completaAlta
- 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)

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:

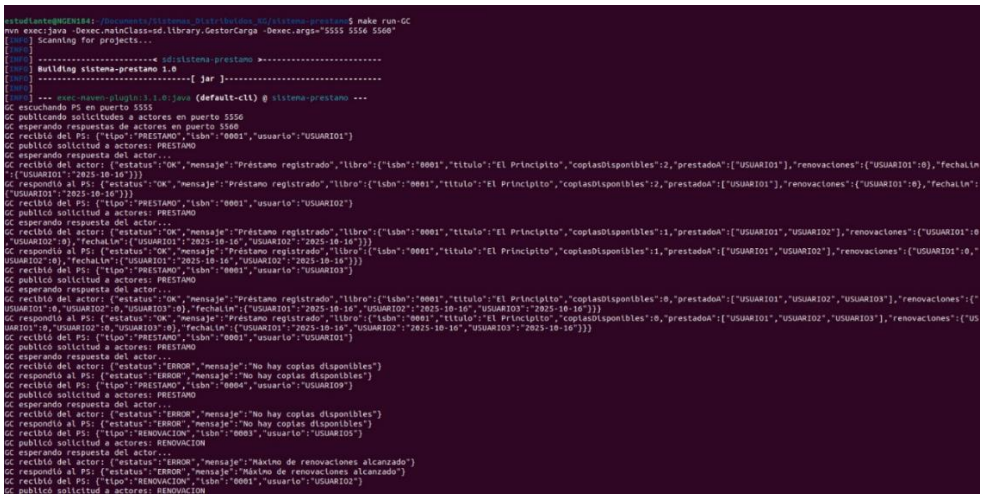
- Baja Carga: 6.38ms - Alta variabilidad inicial
- Alta Carga: 1.83ms - Mejora significativa bajo carga
- Interpretación: El sistema se estabiliza cuando todos los componentes operan sincrónicamente

Análisis Comparativo:

Consistencia Baja Carga: Sistema Original (0.79ms vs 6.38ms)
Consistencia Alta Carga: Sistema Síncrono (1.83ms vs 1.50ms)

# 8 Resultados de Pruebas de Desempeño

## 8.1 Componentes en Ejecución

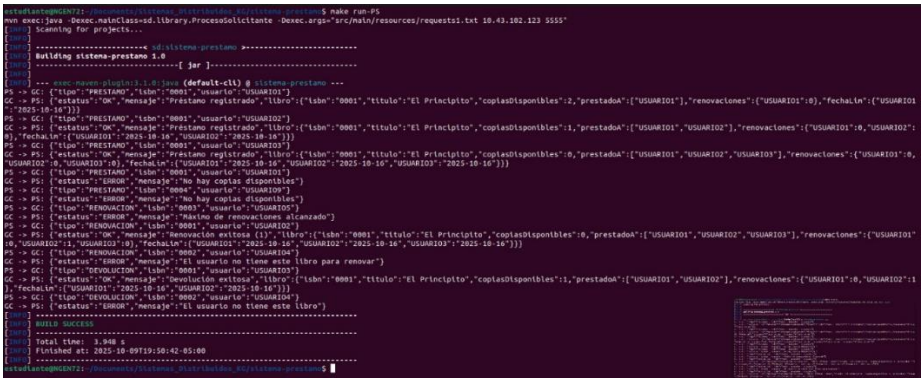




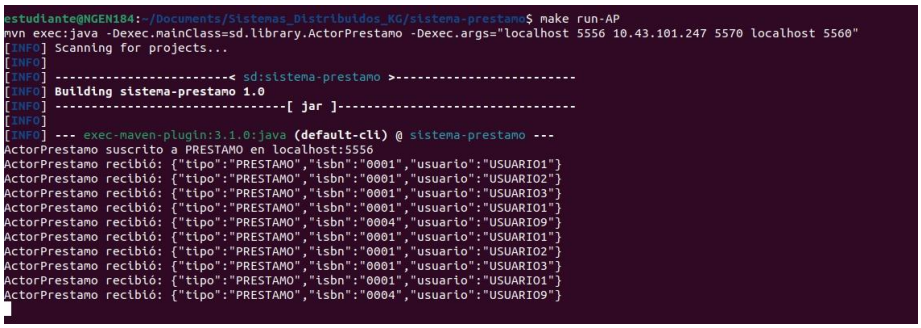
**Descripción:** Gestor de Carga procesando solicitudes.

Consola del Gestor de Carga (GC) en ejecución, mostrando el procesamiento de solicitudes entrantes y la coordinación de operaciones entre los diferentes actores y gestores de almacenamiento.

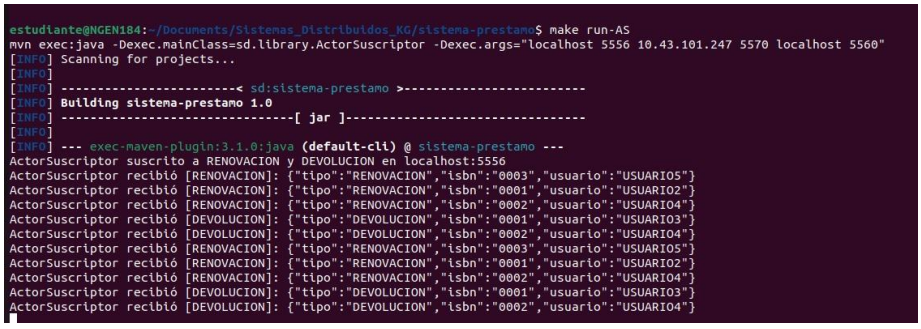
Vista detallada del Gestor de Carga durante una prueba de carga, donde se observan los logs de las solicitudes recibidas y las respuestas enviadas a los Procesos Solicitantes.



**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 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.

# 10 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.