

Diseño y Construcción de una Red Social Distribuida

Aplicación de los Conceptos de Sistemas Distribuidos Estudiados en Clase

Victor Pacheco Jose Agustín Del Toro González

28 de noviembre de 2025

Resumen

Este informe describe el diseño, análisis y justificación teórica del proyecto “Red Social Distribuida”. Se presenta la arquitectura final basada en microservicios, las decisiones de diseño distribuidas, y cómo cada componente del sistema aplica directamente los conceptos estudiados en las conferencias: arquitecturas, procesos, comunicación, coordinación, consistencia, replicación, tolerancia a fallos, cache, privacidad y escalabilidad geográfica.

Índice

1. Introducción	3
2. Arquitectura Distribuida del Sistema	3
2.1. Microservicios	3
2.2. Sharding	3
3. Procesos y Modelo de Ejecución	3
4. Comunicación Distribuida	3
4.1. Comunicación Sincrónica: HTTP/REST	4
4.2. Comunicación Asincrónica Persistente: RabbitMQ	4
5. Nombrado y Direccionamiento	4
6. Coordinación y Consenso	4
6.1. Elección de Líder	4
6.2. Coordinación entre Réplicas	5
7. Consistencia	5

8. Tolerancia a Fallos	5
8.1. Redundancia Física	5
8.2. Redundancia de Información	5
8.3. Mecanismos de Recuperación	5
8.4. Disponibilidad y Confiabilidad	5
9. Cache Distribuida	5
10.Privacidad Distribuida	6
11.Conclusión	6

1. Introducción

El presente proyecto consiste en diseñar e implementar una **red social distribuida** capaz de soportar usuarios, posts, relaciones (follow/unfollow), autenticación distribuida.

El objetivo de este informe es demostrar **cómo cada concepto estudiado en el curso se aplica directamente en el diseño del proyecto**.

2. Arquitectura Distribuida del Sistema

2.1. Microservicios

La arquitectura escogida es una **arquitectura orientada a servicios (SOA)** basada en microservicios,

Los servicios definidos son:

- **Auth-Service**: manejo de autenticación distribuida y emisión de tokens JWT.
- **User-Service**: gestión de perfiles y privacidad.
- **Relation-Service**: implementación del grafo distribuido de seguidores.
- **Post-Service**: creación y consulta de publicaciones.
- **Feed-Service**: generación del feed mediante mensajes asincrónicos.
- **API-Gateway**: punto único de entrada que resuelve el nombrado y direccionamiento.

2.2. Sharding

El sistema distribuye usuarios, posts y relaciones entre múltiples bases de datos mediante particionamiento horizontal por rango o hash. Este mecanismo:

- Reduce la carga.
- Permite escalabilidad independiente.
- Evita un único punto de fallo.

3. Procesos y Modelo de Ejecución

Cada microservicio corre como un **proceso independiente**.

Además, el Feed-Service utiliza múltiples workers que actúan como **procesos concurrentes** que reciben mensajes desde RabbitMQ, siguiendo el paradigma de *message passing*.

4. Comunicación Distribuida

Se aplican los siguientes modelos:

4.1. Comunicación Sincrónica: HTTP/REST

La comunicación directa entre microservicios (ej., User-Service → Relation-Service) ocurre mediante solicitudes HTTP:

- No persistente.
- Sincrónica.
- Transparente.

Corresponde a un modelo RPC simplificado.

4.2. Comunicación Asincrónica Persistente: RabbitMQ

La creación de posts se comunica al Feed-Service mediante un **broker de mensajes**. Esto aplica:

- Mensajería persistente.
- Desacoplamiento temporal.
- Resiliencia ante fallos.

Este mecanismo implementa completamente la teoría de *Message-Oriented Middleware*.

5. Nombrado y Direccionamiento

La arquitectura utiliza:

- **Nombrado estructurado:** el API-Gateway actúa como un DNS interno.
- **Nombrado plano distribuido:** el sharding basado en hash distribuye entidades sin jerarquía.
- **Descubrimiento de servicios:** cada servicio se registra mediante rutas lógicas.

6. Coordinación y Consenso

El sistema implementa coordinación mediante:

6.1. Elección de Líder

Cada shard puede poseer múltiples réplicas. Se implementa una elección de líder estilo Bully/Raft simplificado:

- La réplica con mayor prioridad actúa como líder.
- Si el líder falla, las réplicas realizan una elección.

6.2. Coordinación entre Rélicas

El líder recibe escrituras, las propaga a las rélicas y garantiza orden.

7. Consistencia

El sistema aplica múltiples modelos de consistencia según el contexto:

- **Consistencia eventual:** el feed se actualiza de forma asíncrona.
- **Consistencia causal:** si A sigue a B y luego B publica, A debe ver esa publicación.
- **Lecturas monótonas:** cada usuario siempre ve su propio feed de forma creciente.
- **Lee tus escrituras:** los cambios de perfil son visibles inmediatamente al propio usuario.

8. Tolerancia a Fallos

Se aplican directamente los conceptos de *Tolerancia a Fallos*:

8.1. Redundancia Física

Cada shard posee múltiples rélicas.

8.2. Redundancia de Información

RabbitMQ almacena los mensajes en disco hasta que los consumidores los procesan.

8.3. Mecanismos de Recuperación

- Recuperación de rélicas mediante sincronización con la copia primaria.
- Offset de mensajes para evitar reprocesamiento.

8.4. Disponibilidad y Confiabilidad

El uso de microservicios evita fallos globales.

9. Cache Distribuida

Redis proporciona:

- Cache por región geográfica.
- Reducción de latencia en el Feed-Service.
- Vistas consistentes mediante expiración controlada.

10. Privacidad Distribuida

El User-Service controla políticas de privacidad, y el Feed-Service verifica permisos antes de servir el contenido. Esto se logra mediante:

- JWT para identidad distribuida.
- Control de acceso descentralizado.
- Consistencia fuerte para reglas de privacidad.

11. Conclusión

El proyecto implementa una arquitectura distribuida completamente alineada con los conceptos teóricos estudiados en el curso. Cada decisión —arquitectura, comunicación, consistencia, coordinación, tolerancia a fallos y privacidad— se fundamenta en principios formales de sistemas distribuidos. El resultado es una red social escalable, resiliente, segura y distribuida.