Sistema de Scrapping Web Distribuido

Proyecto de Sistemas Distribuidos

Implementación de un sistema distribuido para extracción automatizada de contenido web utilizando Docker Swarm

Estudiantes: David Sánchez Iglesias y Manuel Alejandro Gamboa Hernández
Asignatura: Sistemas Distribuidos

Universidad de La Habana

1 de diciembre de 2025

Índice

1 Introducción

1.1 Contexto del Proyecto

Los sistemas distribuidos han revolucionado la manera en que procesamos y analizamos grandes volúmenes de información en la era digital. En particular, el web scrapping o extracción automatizada de contenido web se ha convertido en una técnica fundamental para la recopilación de datos a gran escala, utilizada en campos que van desde el análisis de mercado hasta la investigación académica.

El presente proyecto implementa un **Sistema de Scrapping Web Distribuido** que aprovecha las ventajas de la computación distribuida para realizar extracción de contenido web de manera eficiente, escalable y tolerante a fallos. El sistema está diseñado bajo los principios fundamentales de los sistemas distribuidos: distribución de carga, escalabilidad horizontal, tolerancia a fallos y comunicación eficiente entre componentes.

1.2 Problemática Abordada

El scrapping web tradicional, ejecutado en un solo servidor, presenta limitaciones significativas:

- **Limitaciones de rendimiento**: Un único nodo solo puede procesar un número limitado de páginas web simultáneamente
- **Falta de escalabilidad**: No es posible aumentar la capacidad de procesamiento dinámicamente
- **Punto único de falla**: El sistema completo se ve afectado si el servidor único falla
- **Ineficiencia en recursos**: No aprovecha completamente los recursos disponibles en múltiples máquinas

1.3 Objetivos del Sistema

1.3.1. Objetivo General

Desarrollar un sistema distribuido que permita realizar scrapping web de manera eficiente, escalable y confiable, utilizando múltiples nodos de procesamiento coordinados a través de una arquitectura cliente-servidor distribuida.

1.3.2. Objetivos Específicos

1. **Especialización de nodos**: Implementar tres tipos de nodos especializados (Base de Datos, Enrutador, Scrapper) con responsabilidades claramente definidas
2. **Pipeline de datos eficiente**: Desarrollar un flujo de datos optimizado desde la entrada de URLs hasta la persistencia de resultados
3. **Workers puros escalables**: Diseñar nodos scrapper stateless que permitan escalabilidad horizontal sin complejidad adicional
4. **Coordinación inteligente**: Implementar nodos enrutadores que gestionen eficientemente la distribución de tareas y recolección de resultados

5. **Persistencia confiable**: Desarrollar nodos de base de datos que garanticen el almacenamiento seguro de información scrapeada
6. **Sistema de liderazgo distribuido**: Implementar algoritmo bully para elección automática de líderes en cada grupo de nodos
7. **Tolerancia a fallos multi-nivel**: Desarrollar mecanismos de recuperación que incluyan IP caché para fallos de DNS y elección automática de líderes
8. **Replicación automática**: Garantizar que la información se replique automáticamente entre nodos del mismo grupo
9. **Filtrado de calidad**: Asegurar que solo datos exitosamente scrapeados lleguen a la capa de persistencia

1.4 Arquitectura General del Sistema

El sistema implementa una arquitectura distribuida de tres capas con especialización de roles. La arquitectura está compuesta por tres tipos de nodos principales que operan de manera coordinada:

1.4.1. Nodos Base de Datos

Función: Almacenamiento y persistencia de información

- Reciben datos scrapeados procesados desde los nodos enrutadores
- Gestionan el almacenamiento persistente de la información extraída
- Proveen servicios de consulta y recuperación de datos históricos
- Implementan mecanismos de redundancia y respaldo de información

1.4.2. Nodos Enrutador/Coordinador

Función: Transmisión de información y manejo de peticiones

- Actúan como intermediarios de comunicación entre diferentes tipos de nodos
- Transmiten información entre los líderes de los diferentes grupos
- Manejan las peticiones de los clientes externos
- Facilitan la comunicación inter-grupos sin distribuir tareas directamente

1.4.3. Nodos Scrapper (Workers)

Función: Procesamiento puro de extracción web

- Workers especializados en extracción de contenido web
- Reciben URLs específicas para procesar
- Ejecutan el proceso de scrapping de manera autónoma
- Retornan resultados estructurados al nodo enrutador

1.4.4. Infraestructura de Soporte

- **Red Overlay**: Infraestructura de comunicación basada en Docker Swarm
- **Sistema de Descubrimiento**: Mecanismo automático para detección y registro de servicios
- **Gateway API**: Interfaz REST para interacción externa y monitoreo del sistema

2 Sistema de Balanceo de Carga y Liderazgo Distribuido

2.1 Arquitectura Común de Comunicación

Todos los tipos de nodos del sistema (scrapper, enrutador y base de datos) comparten una arquitectura común de comunicación que implementa cuatro actividades fundamentales:

2.1.1. Sistema de Escucha

- Apertura de puerto de escucha mediante `socket.bind ()`
- Hilo dedicado para `socket.accept ()` que registra nuevos sockets
- Creación de hilos especializados para procesar mensajes entrantes

2.1.2. Conexión con Nodos Semejantes

- Método `buscar_semejantes ()` para conectarse con nodos del mismo rol
- Uso de `-hostname` y `-network-alias` de Docker
- `socket.getaddrinfo (network_alias)` para obtener IPs de servicios
- Establecimiento de conexiones peer-to-peer mediante `socket.create_connection ()`
- Ejecución periódica para descubrir nuevos servicios dinámicamente

2.1.3. Sistema de Envío de Mensajes

- Cola de mensajes con tuplas (mensaje, socket_destino)
- Proceso `send_worker ()` que consume la cola con timeout
- Protocolo de envío: longitud del mensaje + mensaje completo
- Método thread-safe para inserción en la cola

2.1.4. Funcionalidades de Mantenimiento

- `send_heartbeat ()`: Confirmación periódica de conectividad
- Monitoreo del estado de conexiones peer-to-peer
- Detección de fallos y reconexión automática
- Gestión del ciclo de vida de hilos de comunicación

2.2 Algoritmo Bully para Elección de Líderes

2.2.1. Principio del Algoritmo

El sistema implementa el algoritmo bully para la elección de líderes en cada grupo de nodos:

- **Criterio de selección**: El nodo con la IP más alta se convierte en líder
- **Proceso determinístico**: No requiere votación ni intervención manual
- **Automático**: La conexión peer-to-peer facilita la comparación de IPs
- **Jerárquico**: Basado en un orden natural (direcciones IP)

2.2.2. Proceso de Elección

1. Cada nodo obtiene las IPs de todos los nodos del mismo tipo
2. Se establece comunicación peer-to-peer con los nodos descubiertos
3. Comparación automática de IPs durante el proceso de conexión
4. El nodo con mayor IP asume automáticamente el rol de líder
5. Notificación implícita del liderazgo a través de las conexiones

2.2.3. Responsabilidades Diferenciadas por Tipo de Nodo

Líderes de Base de Datos y Enrutadores:

- Replican toda información recibida a sus subordinados
- Mantienen sincronización completa de datos en el grupo
- Coordinación inter-grupos con otros líderes
- Gestión de replicación y consistencia de información

Líder de Scrappers:

- **No replica información** — mantiene función pura de procesamiento
- Gestiona distribución de carga entre scrappers subordinados
- Reporta estado del grupo sin distribuir datos de scrapping

2.3 Recuperación ante Fallos de Liderazgo

2.3.1. Detección de Fallos

- Monitoreo mediante heartbeat entre nodos
- Timeout de comunicación para detectar nodos caídos
- Verificación periódica del estado de conexiones

2.3.2. Reelección Automática

Cuando el líder actual falla:

1. Los nodos detectan la pérdida de comunicación con el líder
2. Reinicio del proceso de elección bully
3. El nodo con la segunda IP más alta asume el liderazgo
4. Reestablecimiento de conexiones inter-grupos
5. Continuidad operativa sin intervención manual

2.3.3. Garantías del Sistema

- **Disponibilidad**: Reelección automática sin interrupción de servicio
- **Consistencia**: Un único líder por grupo en todo momento
- **Determinismo**: Resultado predecible basado en IPs
- **Escalabilidad**: Incorporación transparente de nuevos nodos

3 Componente Scrapper: Nodo Worker Especializado

3.1 Descripción General

El componente **Scrapper** constituye la capa de procesamiento del sistema distribuido, organizado jerárquicamente con un **líder que coordina y distribuye tareas** entre los **nodos subordinados**. La función del líder es recibir URLs de los enrutadores y distribuirlas entre sus workers subordinados, además de, como sus subordinados, ejecutar el scrapping web y retornar resultados estructurados.

3.1.1. Principios de Diseño del Worker con Coordinación

- **Especialización funcional**: Se enfoca únicamente en la extracción de contenido web
- **Worker puro**: No replica información, solo procesa URLs (URL $\rightarrow$ Resultado)
- **Participación en liderazgo**: Vota por líder y acepta coordinación del grupo
- **Stateless para datos**: No mantiene estado de scrapping entre tareas

- **Escalabilidad horizontal**: Permite instanciación múltiple con coordinación automática

Cada instancia de scrapper participa en el sistema de liderazgo distribuido de su grupo. El líder scrapper actúa como coordinador con otros grupos, y transmite las direcciones de los otros líderes con los que interactúa a sus subordinados en caso de cambios en la topología o fallos detectados.

3.2 Arquitectura del Scrapper

3.2.1. Diseño de Clases

El scrapper está implementado a través de la clase `ScrapperNode`, que encapsula toda la funcionalidad necesaria para:

- Autodescubrimiento de servicios coordinadores
- Gestión de conexiones de red persistentes
- Procesamiento asíncrono de tareas de scrapping
- Comunicación bidireccional con el coordinador
- Manejo de estados y recuperación ante fallos

3.2.2. Patrones de Diseño Implementados

1. Patrón Observer: Para el manejo de eventos de red y cambios de estado
2. Patrón Producer-Consumer: Para la gestión de colas de mensajes entre hilos
3. Patrón State Machine: Para el manejo de estados del scrapper (disponible, ocupado, desconectado)

3.3 Funcionalidades Principales

3.3.1. Gestión de Conexiones

El sistema implementa un modelo de conexiones persistentes con gestión de hilos especializada, utilizando protocolos binarios optimizados y colas de mensajes thread-safe.
Características de la gestión de conexiones:

- **Protocolo binario optimizado**: Uso de longitud prefijada para eficiencia
- **Cola de mensajes thread-safe**: Comunicación asíncrona entre hilos
- **Reconexión automática**: Detección y recuperación de desconexiones
- **Timeouts configurables**: Prevención de bloqueos indefinidos

3.3.2. Procesamiento de Tareas de Scrapping

El núcleo del procesamiento de tareas integra el módulo especializado `scrapper.py`, que implementa el scrapping web utilizando librerías como `requests` y `BeautifulSoup`.

3.3.3. Sistema de Heartbeat y Monitoreo

Implementación de un sistema de latidos para mantener la conectividad mediante señales periódicas cada 60 segundos que incluyen identificación del cliente y timestamp para monitoreo de conectividad.

3.4 Ventajas del Diseño Distribuido del Scrapper

3.4.1. Escalabilidad Horizontal

- Capacidad de agregar nodos scrapper dinámicamente
- Distribución automática de carga entre nodos disponibles
- Adaptación automática a variaciones en la demanda

3.4.2. Tolerancia a Fallos

- Detección automática de fallos de nodo
- Redistribución de tareas ante fallos
- Reconexión automática de nodos recuperados

3.4.3. Eficiencia de Recursos

- Procesamiento paralelo de múltiples URLs
- Utilización optimizada de recursos de red
- Balanceo automático de carga de trabajo

4 Componente Enrutador: Nodo de Comunicación y Gestión de Peticiones

4.1 Descripción General

Los nodos enrutadores constituyen la capa de comunicación del sistema distribuido, actuando como intermediarios que gestionan las peticiones de los clientes y facilitan la comunicación entre los diferentes grupos de nodos. Su función principal es recibir consultas sobre información almacenada y URLs a procesar, transmitiendo esta información a los líderes correspondientes de cada grupo.

4.1.1. Funciones Principales del Enrutador

- **Gestión de peticiones de clientes**: Reciben consultas sobre datos scrapeados almacenados
- **Recepción de URLs**: Punto de entrada para nuevas tareas de scrapping
- **Transmisión inter-grupos**: Facilitan comunicación entre líderes de diferentes grupos

- **Coordinación de respuestas**: Recolectan y organizan respuestas para los clientes
- **Indexación de respuestas**: Mantienen un índice de URLs procesadas y sus ubicaciones en base de datos, concretamente las IPs de los nodos BD donde se almacenan los datos scrapeados

4.2 Arquitectura del Enrutador

El nodo enrutador utiliza la arquitectura común de comunicación descrita en la sección de balanceo de carga, especializándose en:

- Gestión de peticiones de clientes externos
- Coordinación entre líderes de diferentes grupos
- Distribución de URLs de scrapping a procesar
- Recolección y organización de respuestas

4.2.1. Responsabilidades del Líder Enrutador

- Establecimiento de conexión socket con líder de BD
- Búsqueda activa de servicios de otros roles
- Envío de URLs a procesar al líder del grupo scrapper
- Proporciona IP del líder BD para almacenamiento directo
- Coordina el flujo de información entre grupos

5 Componente Base de Datos: Nodo de Almacenamiento y Persistencia

5.1 Descripción General

Los nodos de base de datos constituyen la capa de persistencia del sistema, encargados de almacenar la información extraída por los scrappers y gestionar la replicación de datos para garantizar disponibilidad y consistencia. Implementan un sistema de liderazgo distribuido con replicación aleatoria y gestión de logs para recuperación ante fallos.

5.1.1. Funciones Principales

- **Almacenamiento de datos**: Persistencia de información scrapeada
- **Replicación distribuida**: Almacenamiento en múltiples nodos (típicamente 3 réplicas)
- **Gestión de logs**: Registro de operaciones para recuperación
- **Coordinación de acceso**: Facilitación de consultas de datos almacenados

5.2 Arquitectura de Base de Datos

Los nodos BD utilizan la arquitectura común de comunicación, especializándose en:

5.2.1. Especialización para Almacenamiento

- Recepción directa de datos desde nodos scrapper
- Procesamiento de información estructurada
- Gestión de metadatos y logs de operaciones
- Coordinación de replicación entre nodos BD

5.2.2. Responsabilidades del Líder BD

- **Gestión de replicación**: Selección aleatoria de 3 nodos BD para almacenamiento
- **Coordinación con enrutador**: Envío de tuplas (URL, IPs) con ubicaciones de datos
- **Gestión de logs**: Almacenamiento de logs de todos los nodos no-líderes
- **Recuperación ante fallos**: Reconstrucción de nodos caídos mediante logs

5.3 Estrategia de Replicación y Recuperación

5.3.1. Replicación Aleatoria

Proceso de Almacenamiento:

1. Recepción de información desde scrapper
2. Selección aleatoria de 3 nodos BD para replicación
3. Distribución de datos a nodos seleccionados
4. Envío de tupla (URL, IPs_almacenamiento) al líder enrutador
5. Registro de operación en logs para trazabilidad

5.3.2. Recuperación ante Fallos

Caída de Nodo No-Líder:

1. Detección de fallo mediante ausencia de heartbeat
2. Incorporación de nuevo nodo BD al grupo
3. Reconstrucción usando logs almacenados por el líder
4. Restauración de réplicas y mantenimiento de factor de replicación

Caída del Líder BD:

1. Elección automática de nuevo líder (algoritmo bully)

2. Envío de logs locales al nuevo líder por parte de todos los nodos
3. Consolidación de información de logs distribuidos
4. Restablecimiento de función de liderazgo y coordinación
5. Continuidad de replicación usando logs históricos

6 Flujo de Datos y Comunicación Inter-Nodos

6.1 Pipeline de Procesamiento Distribuido

El sistema implementa un pipeline de datos distribuido que sigue el siguiente flujo:

1. **Entrada de URLs**: Las URLs a procesar ingresan al sistema a través del Gateway API
2. **Recepción**: Los nodos enrutadores reciben las peticiones y las transmiten al líder correspondiente
3. **Distribución de Tareas**: Los **líderes de cada grupo** distribuyen las tareas entre sus nodos subordinados
4. **Procesamiento**: Los nodos scrapper ejecutan el scrapping de manera independiente bajo coordinación de su líder
5. **Recolección**: Los líderes recolectan resultados y coordinan con otros líderes
6. **Transmisión**: Los enrutadores transmiten la información entre líderes de diferentes grupos
7. **Persistencia**: Los datos procesados llegan al líder de BD para distribución y almacenamiento

6.2 Patrones de Comunicación

6.2.1. Enrutador → Líder Scrapper

- **Tipo**: Transmisión de información entre líderes
- **Contenido**: URLs recibidas de clientes
- **Protocolo**: Comunicación líder a líder

6.2.2. Líder Scrapper → Subordinados Scrapper

- **Tipo**: Distribución real de tareas dentro del grupo
- **Contenido**: URLs específicas asignadas por el líder
- **Responsable**: El líder scrapper gestiona la distribución

6.2.3. Scrapper → Líder Base de Datos

- **Tipo**: Envío directo de resultados
- **Contenido**: Datos extraídos procesados
- **Optimización**: Evita doble salto de información

6.3 Tolerancia a Fallos y Mecanismos de Recuperación

6.3.1. Estrategias de Tolerancia a Fallos por Capas

Nivel de Pipeline:

- **Timeout de scrapping**: Los workers tienen límites de tiempo por tarea
- **Reasignación automática**: Las tareas fallidas se reasignan a otros workers
- **Filtrado de resultados**: Solo los scrappings exitosos continúan en el pipeline
- **Heartbeat continuo**: Detección temprana de nodos no responsivos

6.3.2. IP caché para Recuperación de DNS

Como medida adicional de tolerancia a fallos, cada nodo implementa un **sistema de caché de IPs** que proporciona resiliencia ante fallos del servicio DNS de Docker:

Características del IP caché:

- **Registro automático**: Cada nodo mantiene un registro de IPs de nodos con los que ha interactuado previamente
- **Recuperación ante fallos de DNS**: Si el servicio DNS de Docker falla, los nodos pueden continuar comunicándose usando las IPs cacheadas
- **Persistencia local**: El caché se mantiene localmente en cada nodo para acceso rápido
- **Actualización dinámica**: Las IPs se actualizan automáticamente durante las comunicaciones exitosas

Ventajas del IP caché:

- Continuidad operativa ante fallos de infraestructura
- Reducción de dependencias externas críticas
- Mejora en los tiempos de recuperación del sistema
- Mantenimiento de conectividad entre nodos conocidos

7 Consistencia y Replicación de Datos

7.1 Estrategias de Distribución de Datos

7.1.1. Modelo de Replicación por Grupos

El sistema implementa diferentes estrategias de replicación según el tipo de nodo:

- **Grupos de Base de Datos**: Replicación completa (Full Replication)
  - Cada nodo BD mantiene copia completa de todos los datos
  - Sincronización inmediata tras cada escritura
  - Consistencia fuerte dentro del grupo
- **Grupos de Coordinadores**: Replicación de estado
  - Estado del sistema y decisiones replicadas
  - Información de nodos activos sincronizada
  - Configuraciones y políticas distribuidas
- **Grupos de Scrappers**: Sin replicación de datos
  - Cada scrapper mantiene solo su estado local
  - Resultados enviados directamente a coordinadores
  - Funcionalidad pura sin persistencia

7.1.2. Cantidad y Distribución de Réplicas

Política de Replicación:

- **Factor de Replicación**: Mínimo 3 réplicas por dato crítico
- **Distribución Geográfica**: Réplicas en diferentes nodos físicos
- **Algoritmo de Distribución**: **Selección aleatoria** de nodos destino
- **Simplicidad**: Sin balanceado de carga complejo por ahora

El sistema implementa replicación mediante selección aleatoria de nodos disponibles, manteniendo un mínimo de 3 réplicas por dato crítico cuando sea posible.

7.2 Confiabilidad de Réplicas tras Actualizaciones

7.2.1. Protocolo de Consistencia

Garantías de Consistencia:

- **Consistencia Eventual**: Para datos no críticos
- **Consistencia Fuerte**: Para datos del sistema y configuraciones
- **Ordenamiento de Escrituras**: Timestamps para resolver conflictos

Mecanismos de Verificación:

- **Checksums**: Verificación de integridad de datos replicados
- **Heartbeat con Estado**: Inclusión de información de consistencia
- **Reconciliación Periódica**: Comparación y corrección de diferencias

7.2.2. Recuperación ante Inconsistencias

1. **Detección**: Identificación de réplicas inconsistentes
2. **Votación**: Determinación de versión correcta mediante consenso
3. **Corrección**: Actualización de réplicas incorrectas
4. **Verificación**: Confirmación de consistencia restaurada

8 Conclusiones Parciales

La arquitectura de tres capas con sistema de liderazgo distribuido proporciona una solución robusta y escalable para el procesamiento distribuido de web scrapping. La combinación del algoritmo bully para elección de líderes, la replicación automática de información y el sistema de IP caché para tolerancia a fallos de DNS, resulta en un sistema altamente resiliente.

El diseño de microservicios puros con threading multi-propósito optimiza el uso de recursos mientras mantiene separación clara de responsabilidades. La distribución en redes Docker overlay facilita el escalado horizontal y la gestión de servicios distribuidos geográficamente.

El sistema de replicación diferenciado por grupos — con replicación completa para datos críticos y especialización sin replicación para workers — proporciona un balance óptimo entre consistencia, rendimiento y escalabilidad. Los mecanismos de consistencia eventual y fuerte, según el tipo de datos, garantizan la integridad del sistema.

El diseño jerárquico con líderes en todos los grupos optimiza la comunicación intergrupos mientras mantiene redundancia y capacidad de recuperación automática ante fallos. Los nodos scrapper, aunque participan en el sistema de liderazgo distribuido, mantienen su especialización como workers puros, garantizando simplicidad operativa y máxima escalabilidad horizontal con coordinación inteligente.

Las características implementadas, como el autodescubrimiento de servicios mediante DNS de Docker, la gestión de conexiones persistentes con threading especializado, el sistema de heartbeat distribuido y las estrategias de replicación adaptativas, demuestran la aplicación integral de conceptos fundamentales de sistemas distribuidos en un contexto real de procesamiento de datos web.