Sistema de Scrapping Web Distribuido

Proyecto de Sistemas Distribuidos

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

Estudiante: [Nombre del Estudiante]
Matrícula: [Número de Matrícula]
Profesor: [Nombre del Profesor]
Asignatura: Sistemas Distribuidos

Universidad [Nombre de la Universidad]

29 de noviembre 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 solo nodo 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
- **Limitaciones de red**: Una sola IP puede ser bloqueada por políticas anti-scrapping

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 democrática de líderes en cada grupo de nodos
7. **Tolerancia a fallos multi-nivel**: Desarrollar mecanismos de recuperación que incluyan IP cache 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 basada en el patrón **Producer-Consumer** distribuido 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: Coordinación y distribución de tareas

- Actúan como intermediarios entre la entrada de URLs y los workers
- Distribuyen tareas de scrapping entre nodos scrapper disponibles
- Recolectan resultados de scrapping de los nodos worker
- Envían datos procesados a los nodos de base de datos
- Gestionan el balanceado de carga y tolerancia a fallos

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
- Operan de forma stateless para máxima escalabilidad

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 Componente Scrapper: Nodo Worker Especializado

2.1 Descripción General

El componente **Scrapper** constituye la capa de procesamiento del sistema distribuido, implementado como un **nodo worker puro** y especializado. Su función es estrictamente operativa: recibir una URL específica, ejecutar el proceso de scrapping web y retornar el resultado estructurado.

2.1.1. 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 → Resultado)
- **Participación en liderazgo**: Vota por líder y acepta coordinación del grupo
- **Stateless para datos**: No mantiene estado de scrapping entre tareas
- **Coordinación sin replicación**: Participa en elección de líder pero no distribuye información
- **Escalabilidad horizontal**: Permite instanciación múltiple con coordinación automática

Cada instancia de scrapper participa en el sistema de liderazgo distribuido de su grupo, votando por líderes y aceptando coordinación, pero mantiene su especialización como worker puro. El líder scrapper actúa como coordinador con otros grupos, pero **no replica información de scrapping** a los subordinados, manteniendo la simplicidad operativa.

2.2 Arquitectura del Scrapper

2.2.1. 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

2.2.2. 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)

2.3 Funcionalidades Principales

2.3.1. Autodescubrimiento de Servicios

El scrapper implementa un mecanismo de autodescubrimiento que permite detectar automáticamente coordinadores disponibles en la red:

```
def listen_for_broadcasts(self):
    """Escucha señales de broadcast de los coordinadores"""
    broadcast_socket = socket.socket(socket.AF_INET, socket.
       SOCK_DGRAM)
    broadcast_socket.bind(('', self.broadcast_port))

    while not self.connected:
        data, addr = broadcast_socket.recvfrom(1024)
        message = json.loads(data.decode())

        if message.get('type') == 'coordinator_discovery':
            coordinator_host = message.get('coordinator_host')
            coordinator_port = message.get('coordinator_port')

            if self.connect_to_server():
                break
```

Listing 1: Mecanismo de Autodescubrimiento

Ventajas del autodescubrimiento:

- Eliminación de configuración manual de endpoints
- Detección automática de coordinadores en la red

- Adaptabilidad a cambios en la topología de red
- Simplificación del despliegue de nuevos nodos

2.3.2. Gestión de Conexiones

El sistema implementa un modelo de conexiones persistentes con gestión de hilos especializada:

```
def _send_worker(self):
    """Hilo que envía mensajes desde la cola"""
    while self.connected and not self.stop_event.is_set():
        try:
            message = self.message_queue.get(timeout=1.0)
            if message is None:
                break

            # Protocolo de envio con longitud prefijada
            length = len(message)
            self.socket.send(length.to_bytes(2, 'big'))
            self.socket.send(message)

        except queue.Empty:
            continue
        except Exception as e:
            logging.error(f"Error enviando mensaje: {e}")
            self.connected = False
            break
```

Listing 2: Gestión de Conexiones Persistentes

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

2.3.3. Procesamiento de Tareas de Scrapping

El núcleo del procesamiento de tareas integra el módulo especializado `scrapper.py`:

```
def execute_task(self, task_id, task_data):
    """Ejecuta la tarea asignada"""
    logging.info(f"Ejecutando tarea {task_id}: {task_data}")

    # Marcar como ocupado
    self.update_busy_status(True)

    try:
        url = task_data['url']
```

```
10
11          # Realizar scrapping usando módulo especializado
12          scrape_result = get_html_from_url(url)
13
14          # Preparar resultado optimizado
15          result = {
16              'url': scrape_result['url'],
17              'html_length': len(scrape_result['html']),
18              'links_count': len(scrape_result['links']),
19              'links': scrape_result['links'][:10],  # Primeros 10
                    enlaces
20              'status': 'success'
21          }
22
23      except Exception as e:
24          result = {
25              'status': 'error',
26              'error': str(e)
27          }
28      finally:
29          # Marcar como disponible
30          self.update_busy_status(False)
```

Listing 3: Ejecución de Tareas de Scrapping

Optimizaciones implementadas:

- **Gestión de estado automática**: Actualización automática de disponibilidad
- **Resultados optimizados**: Transmisión de metadata en lugar de contenido completo
- **Manejo robusto de errores**: Captura y reporte de fallos sin afectar el sistema
- **Integración modular**: Uso del módulo `scrapper.py` sin modificaciones

2.3.4. Sistema de Heartbeat y Monitoreo

Implementación de un sistema de latidos para mantener la conectividad:

```
1  def send_heartbeat(self):
2      """Envía señal periódica al servidor"""
3      while self.connected:
4          try:
5              heartbeat_msg = {
6                  'type': 'heartbeat',
7                  'client_id': self.client_id,
8                  'time_now': datetime.now().isoformat()
9              }
10
11             self._enqueue_message(heartbeat_msg)
12             time.sleep(60)  # Heartbeat cada 60 segundos
13
14         except Exception:
```

```
15                self.connected = False
```

Listing 4: Sistema de Heartbeat

2.4 Ventajas del Diseño Distribuido del Scrapper

2.4.1. Escalabilidad Horizontal

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

2.4.2. Tolerancia a Fallos

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

2.4.3. Eficiencia de Recursos

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

2.4.4. Flexibilidad de Despliegue

- Despliegue en múltiples máquinas físicas
- Compatibilidad con contenedores Docker
- Integración con orquestadores como Docker Swarm

3 Flujo de Datos y Comunicación Inter-Nodos

3.1 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. **Distribución**: Los nodos enrutadores reciben las URLs y las distribuyen entre nodos scrapper disponibles
3. **Procesamiento**: Los nodos scrapper ejecutan el scrapping de manera independiente
4. **Recolección**: Los nodos enrutadores recolectan los resultados exitosos

5. **Persistencia**: Los datos procesados se envían a los nodos de base de datos para almacenamiento

3.2 Patrones de Comunicación

3.2.1. Enrutador → Scrapper

- **Tipo**: Comunicación síncrona punto a punto
- **Contenido**: URL específica con metadatos de tarea
- **Protocolo**: TCP con heartbeat para detección de fallos

3.2.2. Scrapper → Enrutador

- **Tipo**: Respuesta con resultado de procesamiento
- **Contenido**: Datos extraídos o información de error
- **Condición**: Solo si el scrapping fue exitoso

3.2.3. Enrutador → Base de Datos

- **Tipo**: Almacenamiento asíncrono de resultados
- **Contenido**: Datos estructurados procesados
- **Garantía**: Solo datos exitosamente scrapeados

3.3 Tolerancia a Fallos y Mecanismos de Recuperación

3.3.1. 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

3.3.2. IP Cache 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 Cache:

- **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 cache se mantiene localmente en cada nodo para acceso rápido
- **Actualización dinámica**: Las IPs se actualizan automáticamente durante las comunicaciones exitosas

Ventajas del IP Cache:

- 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

4 Sistema de Liderazgo Distribuido con Algoritmo Bully

4.1 Arquitectura de Grupos Interconectados

El sistema implementa una arquitectura jerárquica donde cada grupo de nodos (exceptuando los nodos scrapper) está completamente interconectado, formando clusters de alta disponibilidad con replicación automática de información.

4.1.1. Principios de Interconexión y Organización

Grupos con Liderazgo Distribuido:

- **Grupo Base de Datos**: Todos los nodos BD interconectados con replicación completa de información
- **Grupo Enrutadores**: Todos los enrutadores interconectados con replicación de estado y tareas
- **Grupo Scrappers**: Organizados con liderazgo pero **sin replicación de información**, solo coordinación

Comportamiento Diferenciado por Grupo:

- **Base de Datos y Enrutadores**: Replicación automática de toda información recibida
- **Scrappers**: Participan en elección de líder y coordinación, pero **no replican información**
- **Función especializada**: Los scrappers mantienen su rol puro de procesamiento de URLs
- **Coordinación sin replicación**: El líder scrapper coordina con otros grupos pero no distribuye datos internamente

4.2 Implementación del Algoritmo Bully

4.2.1. Elección de Líderes en Todos los Grupos

Para asegurar coordinación y organización, **todos los grupos de nodos** (incluyendo Base de Datos, Enrutadores y Scrappers) utilizan el **Algoritmo Bully** para la elección democrática de líderes:

Proceso de Elección:

1. **Detección de fallo**: Los nodos detectan la desconexión del líder actual
2. **Inicio de elecciones**: Cualquier nodo puede iniciar el proceso de elección
3. **Votación distribuida**: Los nodos participan en la votación según el algoritmo bully
4. **Elección del nuevo líder**: El nodo con mayor prioridad se convierte en líder
5. **Anuncio inter-grupos**: El nuevo líder se comunica con otros grupos para anunciar su cargo

4.2.2. Roles y Responsabilidades del Líder por Tipo de Grupo

Funciones Comunes del Líder ("Jefe") en Todos los Grupos:

- **Representante del grupo**: Actúa como único punto de comunicación con otros grupos
- **Comunicación inter-grupos**: Envía y recibe información desde/hacia líderes de otros grupos
- **Coordinación de operaciones**: Mantiene la organización y sincronización dentro del grupo

Responsabilidades Diferenciadas:

- **Líderes de Base de Datos y Enrutadores**:
  - Replican toda información recibida a sus subordinados
  - Mantienen sincronización completa de datos en el grupo
  - Garantizan consistencia de información
- **Líder de Scrappers**:
  - **No replica información** - Los scrappers mantienen su función pura: procesamiento de URLs
  - Coordina disponibilidad de workers con otros grupos
  - Gestiona distribución de carga entre scrappers subordinados
  - Reporta estado del grupo pero no distribuye datos de scrapping

Comunicación Jerárquica:

- **Líder ↔ Líder**: Comunicación directa entre líderes de diferentes grupos
- **Líder → Subordinados**: Replicación de información dentro del grupo y asignación de tareas
- **Subordinados → Líder**: Reportes de estado y solicitudes de coordinación

4.3 Recuperación ante Fallos de Liderazgo

4.3.1. Detección de Fallos del Líder

Mecanismos de Detección:

- **Heartbeat monitoring**: Los subordinados monitorean constantemente al líder
- **Timeout de comunicación**: Fallos detectados por ausencia de respuesta
- **Verificación distribuida**: Múltiples nodos confirman el fallo antes de actuar

4.3.2. Proceso de Recuperación

Secuencia de Recuperación:

1. **Detección consensuada**: Los miembros restantes confirman la desconexión del líder
2. **Llamada a elecciones**: Los nodos supervivientes inician el proceso de elección
3. **Votación por nuevo líder**: Aplicación del algoritmo bully para selección
4. **Anuncio del nuevo líder**: Comunicación del cambio a otros grupos
5. **Reanudación de operaciones**: Continuación de actividades bajo el nuevo liderazgo

Garantías del Sistema:

- **Continuidad operativa**: Las operaciones se reanudan automáticamente
- **Prevención de problemas de coordinación**: El algoritmo bully evita conflictos de liderazgo
- **Transparencia para otros grupos**: El cambio de liderazgo es comunicado externamente
- **Mantenimiento de jerarquía**: La estructura de comunicación se preserva

4.4 Ventajas de la Arquitectura de Liderazgo

4.4.1. Beneficios Operacionales

- **Coordinación eficiente**: Un solo punto de comunicación por grupo reduce la complejidad
- **Escalabilidad de comunicación**: O(n) comunicaciones entre grupos en lugar de $O(n^2)$
- **Consistencia de datos**: La replicación centralizada garantiza sincronización
- **Recuperación rápida**: El algoritmo bully proporciona elección de líder determinística

4.4.2. Tolerancia a Fallos Mejorada

- **Redundancia activa**: Múltiples nodos pueden asumir el liderazgo
- **Elección automática**: Sin intervención manual en la recuperación
- **Preservación de operaciones**: Continuidad garantizada ante fallos de liderazgo
- **Comunicación resiliente**: Respaldo por IP cache ante fallos de DNS

5 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 cache para tolerancia a fallos de DNS, resulta en un sistema altamente resiliente.

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 al no replicar información, garantizando simplicidad operativa y máxima escalabilidad horizontal con coordinación inteligente.

Las características implementadas, como el autodescubrimiento de servicios, la gestión de conexiones persistentes y el sistema de heartbeat, demuestran la aplicación práctica de conceptos fundamentales de sistemas distribuidos en un contexto real de procesamiento de datos web.