Implementación de un Servidor Distribuido de Spotify con Kademlia y Raft

Alex Sanchez Saez, Carlos Manuel González Peña

July 15, 2024

1 Introducción

En el ámbito de los sistemas distribuidos, la implementación de aplicaciones como un servidor distribuido de Spotify plantea desafíos significativos en términos de escalabilidad, tolerancia a fallos y gestión de datos distribuidos. Este trabajo propone una solución integral utilizando el protocolo *Kademlia* para la gestión de nodos y datos, complementado con el algoritmo *Raft* para la selección de líderes y la consistencia de datos críticos, como las listas de reproducción.

2 Descripción del Sistema

El sistema permite que múltiples nodos se unan y autodescubran en la red utilizando técnicas de broadcast y multicast para la gestión inicial de la red. Cada nodo está representado por un objeto `Node` que incluye su identificador único calculado mediante una función hash `SHA-1`, y su dirección IP junto con el puerto de comunicación.

```python
import hashlib

def sha1_hash(data: str) -> int:
    return int(hashlib.sha1(data.encode()).hexdigest(), 16)

class Node:
    def __init__(self, ip: str, port: int, id=None):
        if id is not None:
            self.id = id
        else:
            self.id = sha1_hash(f"{ip}:{port}")
        self.ip = ip
        self.port = port

    def __repr__(self):
        return f"Node({self.id}, {self.ip}, {self.port})"
```

La red Kademlia se utiliza para la gestión eficiente de la tabla de enrutamiento y la ubicación de datos distribuidos. Cada nodo mantiene un conjunto de

`KBucket` para almacenar los nodos conocidos en diferentes rangos de distancia de su propio ID.

```python
from typing import List

class KBucket:
    def __init__(self, range_start: int, range_end: int):
        self.range_start = range_start
        self.range_end = range_end
        self.nodes = []

    def add_node(self, node: Node):
        # Implementación de añadir y remover nodos en el K-bucket
        pass

    def remove_node(self, node: Node):
        # Implementación de remover nodos del K-bucket
        pass

    def get_nodes(self) -> List[Node]:
        # Obtener todos los nodos en el K-bucket
        pass
```

Raft se implementa para garantizar la consistencia de las listas de reproducción en toda la red. Solo un nodo (el líder) coordina las modificaciones a las listas de reproducción, asegurando así la coherencia y la disponibilidad de los datos críticos.

```python
# Implementación de Raft para la elección de líderes y replicación de datos
# (Detalles específicos del código de Raft estarían aquí)
```

3 Implementación de KademliaRpcNode

La clase `KademliaRpcNode` extiende de `RpcNode` e integra funcionalidades de Kademlia y Raft para la gestión de la red y las listas de reproducción. Aquí se muestran fragmentos clave de su implementación:

```python
class KademliaRpcNode(RpcNode):
    def __init__(self, ip: str, port: int):
        super().__init__(ip, port, None)
        self.routing_table = RoutingTable(self.id)
        self.network = KademliaNetwork(self)
        self.database = PlaylistManager()
        self.requested_nodes = {}
        self.file_transfers = {}
        self.values_requests = {}
        self.pings = {}

        self.consensus = RaftNode(
            Node(self.ip, self.port, self.id), self.network, self.routing_table
        )

    # Métodos adicionales como ping, store, find_node, find_value, etc.
```

4 Conclusiones

La combinación de la red *Kademlia* para el descubrimiento de nodos y la gestión de datos distribuidos, junto con *Raft* para la selección de líderes y la consistencia de las listas de reproducción, proporciona una base sólida para aplicaciones multimedia de alta disponibilidad y rendimiento. Este enfoque ofrece escalabilidad, tolerancia a fallos y consistencia de datos críticos, fundamentales para sistemas distribuidos complejos.

Concluimos que este trabajo presenta una arquitectura robusta y eficiente para la implementación de sistemas distribuidos como alternativa viable a plataformas centralizadas tradicionales.