Análisis de la Arquitectura P2P

Cristian David Dávila García

Juan José Gabriel Cañón Díaz

Universidad EAFIT

Tópicos Especiales en Telemática: ST0263 - 1715

Juan Carlos Montoya Mendoza

06 de Septiembre de 2024

Contenido

Introducción

En este documento, exploramos la arquitectura P2P y la comparamos con el modelo Cliente-Servidor, destacando sus diferencias y características clave. Nos enfocaremos en las clasificaciones de las redes P2P, tanto estructuradas como no estructuradas, explicando sus particularidades y los aspectos fundamentales a considerar al diseñar aplicaciones P2P, como la topología, las tablas de enrutamiento, los procesos de entrada y salida de nodos, y los mecanismos de búsqueda. Además, analizaremos tres ejemplos de redes P2P, describiendo en detalle su arquitectura y funcionamiento. Finalmente, abordaremos el concepto de las Tablas Hash Distribuidas (DHT) y su relevancia en el contexto de las redes P2P.

Comparación entre la arquitectura Cliente/Servidor y P2P

1. Cliente/Servidor

La arquitectura cliente-servidor es un modelo de diseño de sistemas donde las tareas y recursos se distribuyen entre dos entidades principales: clientes y servidores. El cliente es la máquina o aplicación que solicita servicios o recursos, mientras que el servidor es el sistema que almacena los recursos y responde a las solicitudes de los clientes. Este modelo centralizado permite una gestión eficiente, ya que el servidor controla los datos y la funcionalidad, mientras que los clientes interactúan con el servidor para realizar operaciones específicas, como consultas o transacciones.

2. P2P

La arquitectura **P2P (peer-to-peer)** es un modelo descentralizado en el que cada nodo o dispositivo conectado a la red actúa simultáneamente como cliente y servidor. En lugar de depender de un servidor central para compartir recursos o datos, los nodos se comunican directamente entre sí, intercambiando información y recursos. Este enfoque distribuye la carga entre todos los participantes, permitiendo una mayor escalabilidad y resistencia a fallos, pero también presenta desafíos en la coordinación y búsqueda de recursos en redes grandes.

3. Diferencias, retos y oportunidades

a. Diferencias

- **Jerarquía en el sistema:** P2P no tiene una jerarquía entre sus nodos, ya que todos los participantes actúan tanto como clientes

como servidores, mientras que en Cliente-Servidor existe una clara jerarquía vertical, donde los clientes dependen de un servidor central para el acceso y provisión de servicios.

- **Escalabilidad:** La escalabilidad en P2P es significativamente mayor, dado que cada nuevo nodo contribuye con recursos al sistema. En contraste, en el modelo Cliente-Servidor, la escalabilidad está limitada por la capacidad del servidor central, que puede convertirse en un cuello de botella a medida que aumentan los usuarios.
- **Tolerancia a fallos:** El sistema P2P tiene una mayor tolerancia a fallos, porque la caída de un nodo no afecta el funcionamiento global. En Cliente-Servidor, si el servidor falla, el sistema completo puede quedar inoperante.
- **Gestión de seguridad:** La gestión de la seguridad es más compleja en P2P, ya que cada nodo debe encargarse de su propia protección. En Cliente-Servidor, la seguridad se concentra en el servidor, lo que simplifica la gestión.
- **Latencia:** En Cliente-Servidor, la latencia puede verse afectada por la carga del servidor y su distancia respecto a los clientes. En P2P, la carga se distribuye, lo que permite buscar el nodo más cercano, reduciendo la latencia.

b. Retos

i. Cliente/servidor

- **Escalabilidad:** A medida que aumentan los clientes o solicitudes en un modelo Cliente-Servidor, la carga sobre el servidor se incrementa, lo que puede generar cuellos de botella y afectar el rendimiento del sistema.
- **Punto único de Falla - Disponibilidad:** En el modelo Cliente-Servidor, si el servidor central falla, el sistema entero se vuelve inaccesible, lo que representa un punto único de fallo y puede comprometer la disponibilidad del servicio.
- **Costo de Infraestructura:** Mantener un servidor poderoso y redundante para asegurar alta disponibilidad en un modelo Cliente-Servidor puede resultar costoso, especialmente cuando se requiere que el sistema esté disponible en todo momento.
- **Seguridad:** En Cliente-Servidor, el servidor central se convierte en un objetivo clave para ataques, lo que puede comprometer la seguridad del sistema si el servidor es vulnerado.

ii. Peer-To-Peer

- **Consistencia de datos:** Mantener la consistencia de datos en un sistema distribuido sin un control centralizado puede ser complejo. La sincronización entre distintos nodos puede resultar difícil y propensa a errores, afectando la integridad de la información.
- **Seguridad y Confianza:** En un sistema P2P, donde no hay un punto central, asegurar el sistema y gestionar la confianza

entre nodos es más complicado. Garantizar que todos los nodos sigan las mismas reglas de seguridad y comportamiento es un desafío.
- **Gestión de Red:** Sin un servidor central, la gestión de red y la organización de recursos en un sistema distribuido pueden volverse caóticas, especialmente a medida que la red crece y se expande.

c. Oportunidades
   i. Cliente/Servidor
      - **Control centralizado:** El modelo Cliente-Servidor ofrece un control centralizado, lo que permite una gestión más sencilla y coherente de la seguridad, almacenamiento de datos y acceso. Esta centralización facilita la administración y supervisión del sistema.
      - **Facilidad de Mantenimiento:** La facilidad de mantenimiento en Cliente-Servidor se destaca porque las actualizaciones y mejoras del sistema pueden ser centralizadas. Esto simplifica la gestión de versiones, la aplicación de parches de seguridad y la implementación de mejoras, ya que todo se realiza desde un único punto.
   ii. Peer-To-Peer
      - **Escalabilidad:** El modelo P2P ofrece una escalabilidad dinámica, ya que cada nuevo nodo añade recursos y capacidad al sistema. Esto permite que la red crezca de manera efectiva con el aumento en la demanda y el número de usuarios.
      - **Resiliencia:** P2P proporciona una resiliencia y robustez superiores, ya que no depende de un único punto de fallo. La red puede seguir operando incluso si algunos nodos fallan, lo que mejora la disponibilidad y la continuidad del servicio.
      - **Distribución de carga:** La distribución de carga en un sistema P2P permite una utilización más eficiente de los recursos. Al compartir la carga entre varios nodos, se reduce la presión sobre cualquier nodo individual y se mejora el rendimiento general del sistema.

Clasificación de las arquitecturas P2P (unstructured & structured)

- Clasificaciones
  - Completamente Descentralizadas (Unstructured):

    todos sus nodos son equivalentes y no existe un servidor central ni un nodo con un rol especial, ambos actúan como cliente y servidor, permitiéndoles

solicitar y proporcionar datos, realizan búsqueda de datos mediante inundación o consultas a sus nodos vecinos y esos vecinos consultan a sus vecinos y así sucesivamente.

- Híbridas (unstructured):

  Combinan elementos de centralización y descentralización, un servidor central o conjunto de servidores manejan tareas específicas, pero la transferencia de archivos se realiza directamente entre los nodos.

- Basadas en DHT (Structured):

  Organizan los nodos y los datos en un espacio clave bien definido. Cada nodo y/o recurso posee una clave única generada. Los recursos se almacenan en los nodos más cercanos a la clave del recurso, lo cual permite localizar un recurso en un número reducido de saltos, en lugar de recorrer toda la red.

- Basadas en Grafos (Structured):

  Los nodos y datos se organizan en un espacio de coordenadas o gráfico, donde los recursos se asignan a ubicaciones específicas en un espacio multidimensional. Cada nodo es responsable de una región específica del espacio y conoce a sus vecinos cercanos, lo que permite las búsquedas más eficientes moviéndose por el gráfico.

- Aspectos Principales
  - Completamente Descentralizadas:
    - **Topología**: Aleatoria, cada nodo puede conectarse con otro nodo de la red, no hay un patrona fijo o predefinido, cada nodo mantiene conexiones con un subconjunto de otros nodos en la red
    - **Búsqueda**: se realiza mediante una técnica llamada flooding, el nodo busca un archivo enviando una consulta a todos sus vecinos, luego los vecinos envían la misma consulta a sus vecinos y así sucesivamente, el proceso continúa hasta que el archivo se encuentre o termine el tiempo de vida para así evitar bucles infinitos.
    - **Entrada y Salida de Nodos**:
      - **Entrada**: un nuevo nodo se une a la red conectándose con uno o más nodos existentes. Esto suele ocurrir de manera aleatoria o a través de un nodo conocido.
      - **Salida**: Si un nodo se desconecta, simplemente deja de participar en la red, no requiere ningún procedimiento especial, pero puede causar problemas si el nodo posee datos o conexiones importantes.
    - **Tabla de Enrutamiento**: No existe tabla de enrutamiento como tal en estas redes. Cada nodo sólo conoce a sus vecinos directos, y las rutas se forman dinámicamente a través del proceso de inundación.

- Híbridas:
  - **Topología**: parcialmente centralizada. Existe un servidor central o conjunto de servidores que actúan como índice global, mientras que los nodos se conectan entre sí para la transferencia de archivos. La red de nodos puede ser descentralizada, pero existe un servidor central que se introduce en la jerarquía.
  - **Búsqueda**: los nodos envían consultas al servidor central, que mantiene un índice de los archivos disponibles en la red. El servidor central devuelve la ubicación de los nodos que contienen el archivo y la transferencia se realiza directamente entre nodos.
  - **Entrada y Salida de Nodos**:
    - **Entrada**: Los nuevos nodos se registran en el servidor central cuando se unen a la red, informando al servidor sobre los archivos que tienen disponibles.
    - **Salida**: Cuando un nodo sale de la red, el servidor central elimina la información del índice sobre los archivos que estaban disponibles en ese nodo.
  - **Tabla de Enrutamiento**: El servidor central es quien maneja la mayor parte de la información de enrutamiento, proporcionando a los nodos las direcciones de otros nodos para realizar las transferencias.
- Basadas en DHT:
  - **Topología**: definida por la estructura de la DHT, los nodos y datos se organizan en un espacio de claves predefinido, por ejemplo un anillo
  - **Búsqueda**: basada en la clave hash del recurso, por ejemplo, en el caso del anillo, se realiza la búsqueda del recurso siguiendo el anillo hasta encontrar el nodo que tiene la clave más cercana al recurso
  - **Entrada y Salida de Nodos**:
    - **Entrada**: un nodo se une al sistema, se le asigna una clave, el nodo debe informarse sobre sus vecinos cercanos y tomar responsabilidad de un rango de claves.
    - **Salida**: Cuando un nodo sale, sus responsabilidades se transfieren a su sucesor y se realiza un ajuste del sistema para mantener la estructura lógica
  - **Tabla de Enrutamiento**: Cada nodo mantiene una tabla de enrutamiento que contiene referencias a otros nodos en la red. La tabla se actualiza dinámicamente a medida que los nodos entran y salen de la red
- Basadas en Grafos:
  - **Topología**: los nodos y los recursos se organizan en un espacio multidimensional o en una estructura gráfica, donde cada nodo es responsable de una región específica del espacio.
  - **Búsqueda**: Se realiza moviéndose por el espacio gráfico en direcciones que acercan la búsqueda al nodo responsable de la clave buscada.

- **Entrada y Salida de Nodos**:
  - **Entrada**: Cuando un nuevo nodo se une, se le asigna una parte de la zona de un nodo existente, las responsabilidades se dividen entre ellos.
  - **Salida**: Cuando un nodo sale, su zona se redistribuye entre sus vecinos para mantener la cobertura del espacio
- **Tablas de Enrutamiento**: Cada nodo mantiene una tabla de enrutamiento con referencias a sus vecinos más cercanos del espacio gráfico. La tabla de enrutamiento se ajusta automáticamente cuando cambia la topología, debido a la entrada o salida de nodos.

Análisis de redes P2P: Napster, BitTorrent & Chord

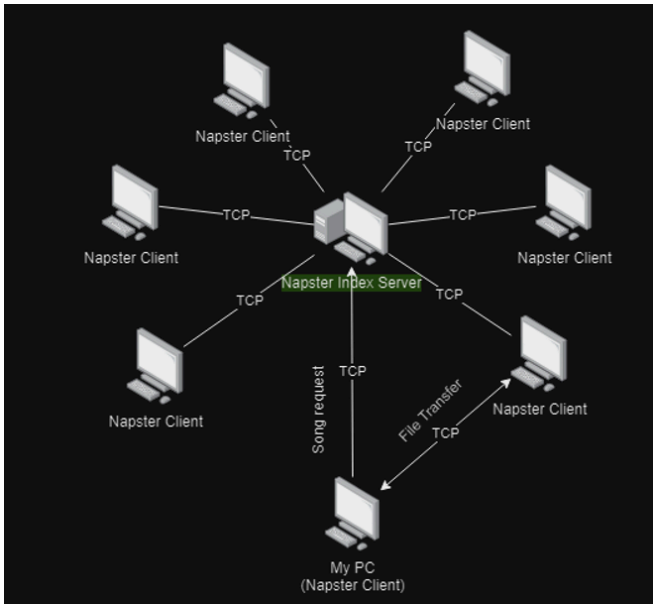
● Napster

Napster fue una de las primeras redes P2P para compartir archivos, pero utilizaba una arquitectura centralizada, donde los servidores desempeñaban un papel importante.

○ Partes y Funcionamiento

- Indexer: Guarda un índice de todos los usuarios de Napster que están actualmente en línea para conectarlos entre ellos. Pero no contiene ni un solo archivo mp4.
- Clientes: Usuarios que comparten o descargan archivos mp3 en el sistema.

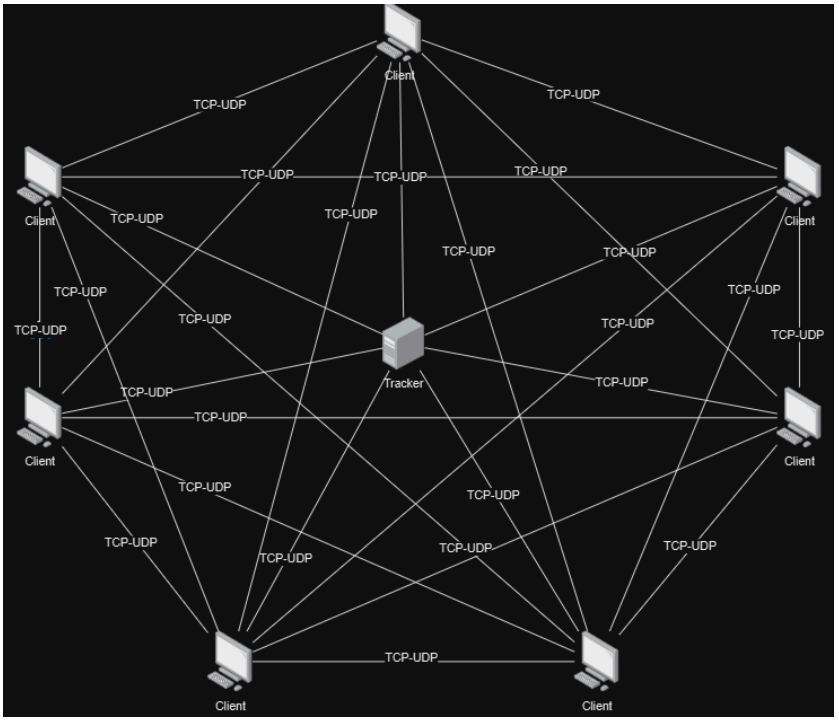
○ Visualización



- BitTorrent

  BitTorrent es una red P2P semi-estructurada diseñada específicamente para compartir archivos grandes de manera eficiente, dividiendo los archivos en pequeñas partes.

  - Partes y Funcionamiento
    - Tracker: provee una lista de archivos disponibles para transferir y permitir al cliente encontrar usuarios peers, conocidos como "seeds", quienes tal vez transfieran los archivos
    - Clientes
      - Seeders: los usuarios que publican un archivo en el sistema
      - Leechers: los usuarios que buscan y descargan archivos de los seeders
  - Visualización

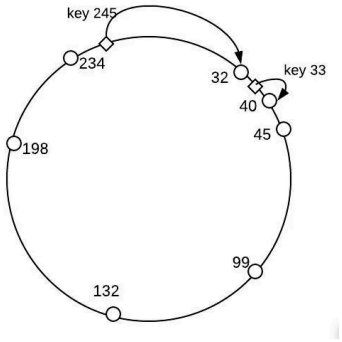


- Chord

  Chord es un sistema P2P estructurado basado en **Distributed Hash Tables (DHT)**, lo que permite una distribución eficiente de los archivos en la red.

  - Partes y Funcionamiento
    - Nodos: Máquinas que participan en la red con identificador único asignado por una función Hash
      - Nodo Sucesor: Es el nodo más cercano en el anillo que sigue de una clave Hash dada, es el nodo responsable de esa clave.

        - Nodo predecesor: Nodo inmediatamente anterior a un nodo en el anillo. Ayuda en la navegación y redistribución de claves cuando un nodo entra o sale.
    - Anillo: Chord organiza los nodos en un anillo lógico, donde las claves y los nodos se colocan en un espacio de identificadores utilizando Hash.
    - Finger Table: Tabla de enrutamiento de cada nodo. Contiene referencias a otros nodos, para permitir que se realicen saltos rápidos para encontrar llaves de una forma más eficiente.
- Visualización



Distributed Hash Tables (DHT)

● Definición de DHT

Una Tabla de Hash Distribuida (DHT) es una estructura de datos distribuida que facilita el almacenamiento y la recuperación eficiente de claves y valores en una red de nodos. Utiliza una función de hash para asignar claves a nodos específicos, donde cada nodo es responsable de un rango de claves y gestiona solicitudes asociadas a esas claves. Esta estructura permite operaciones de búsqueda, inserción y eliminación en tiempo logarítmico, lo que la hace escalable y eficiente en redes grandes, siendo fundamental en aplicaciones de redes peer-to-peer y sistemas distribuidos.

- Utilidad en Redes P2P
    1. **Localización eficiente**: Permiten a los nodos en la red encontrar rápidamente otros nodos que almacenan datos específicos. La DHT organiza la información de manera que cualquier nodo puede buscar y recuperar datos de manera eficiente, incluso en redes grandes y dinámicas.
    2. **Escalabilidad**: A medida que la red crece y se añaden o eliminan nodos, las DHTs mantienen su eficiencia en términos de búsqueda y almacenamiento. Esto es crucial en redes P2P donde los nodos pueden unirse y salir de la red con frecuencia.
    3. **Distribución de Carga**: La carga de almacenamiento y procesamiento se distribuye entre todos los nodos de la red, evitando cuellos de botella y puntos únicos de falla, y mejorando la resiliencia general de la red.

4. **Descentralización**: Eliminan la necesidad de un servidor centralizado, permitiendo que cada nodo actúe de manera autónoma y cooperativa para gestionar y acceder a los datos, lo que es fundamental para la arquitectura descentralizada de las redes P2P.
5. **Robustez**: Al estar diseñada para manejar la entrada y salida dinámica de nodos, una DHT proporciona robustez frente a fallos de nodos y asegura que los datos continúen siendo accesibles incluso si algunos nodos fallan o se desconectan.

Conclusiones

En conclusión, la comparación entre las arquitecturas Cliente-Servidor y Peer-to-Peer (P2P) revela diferencias fundamentales en la distribución y gestión de recursos. El modelo Cliente-Servidor, con su estructura centralizada, facilita el control y la administración de la seguridad y el mantenimiento, pero puede enfrentar problemas de escalabilidad y puntos únicos de falla. En contraste, la arquitectura P2P, al distribuir las funciones de cliente y servidor entre todos los nodos, ofrece una mayor escalabilidad y resiliencia, aunque conlleva desafíos significativos en la gestión de la seguridad y la consistencia de datos.

Las redes P2P se dividen en estructuras centralizadas, híbridas y completamente descentralizadas, cada una con sus propias ventajas y desventajas. Las redes basadas en Tablas Hash Distribuidas (DHT) y grafos estructurados proporcionan soluciones avanzadas para la organización y localización de datos, mejorando la eficiencia y escalabilidad del sistema. Ejemplos como Napster, BitTorrent y Chord ilustran cómo estas arquitecturas se aplican en la práctica, cada una abordando diferentes necesidades y desafíos en el intercambio y distribución de archivos.

Referencias

C, A., V. (2024, 17 julio). BitTorrent: The Engineering behind the BitTorrent protocol. *Medium*. https://medium.com/@abhinavcv007/bittorrent-part-1-the-engineering-behind-the-bittorrent-protocol-04e70ee01d58

Education, T. (2023, 25 diciembre). *Napster: inicio, ascenso y caída del software que revolucionó la industria de la música*. https://thepower.education/blog/napster-software-que-revoluciono-la-industria-de-la-musica

Tanenbaum, A. S., & Van Steen, M. (2001). *Distributed Systems: Principles and Paradigms*. http://ce.miau.ac.ir/azmoninfo/slides.01.pdf

The Lost Packet. (2022, 7 diciembre). *Chord - a distributed hash table* [Vídeo]. YouTube. https://www.youtube.com/watch?v=9kd1aj8E30k

Tyson, J. (2023, 8 marzo). *How the Old Napster Worked*. HowStuffWorks. https://computer.howstuffworks.com/napster.htm

Yang, J. (2021, 14 diciembre). Key lookup in Chord with finger table - Jing Yang - Medium. *Medium*. https://medium.com/@jingyang_56841/key-lookup-in-chord-with-finger-table-c0179bafae13