Tema 3 Sincronización en SSDD

---

1. Introducción

La información relevante se distribuye entre varias máquinas. No existe un reloj común o alguna otra fuente precisa de tiempo global.

Las principales soluciones ante este problema:

- ***Tiempo distribuido***: Cada nodo posee su propio reloj "tiempo físico local", en este caso los relojes son imprecisos, es necesario ajustarlos periódicamente.
- ***Tiempo lógico***: La gestión consistente del estado global requiere al menos ordenar los eventos producidos por los nodos.

2. Tiempo Físico

2.1 Relojes

Los computadores pueden disponer de un reloj físico propio. Estos relojes son dispositivos que cuentan las oscilaciones que ocurren en un cristal "cuarzo" a una frecuencia definida, que normalmente dividen esta cuenta y almacenan el resultado en un registro contador.

2.2 Sesgo y deriva de reloj

Los relojes de los computadores tienden a no estar en un estado perfecto. La diferencia instantánea entre las lecturas de dos relojes cualesquiera se llama ***sesgo***.

La diferencia de tiempo entre el reloj de referencia y el local del computador se denomina ***deriva***.

3. Sincronización de los relojes físicos

Los computadores de un SD[1] poseen relojes que no están sincronizados "derivas de tiempo", por tanto es importante asegurar una buena sincronización en aplicaciones a tiempo real, análisis de rendimiento,..

En los SD encontramos sincronización interna y externa.

[1] SD → Sistema Distribuido

- ***Sincronización Interna*** → Consiste en mantener bien sincronizados entre sí los relojes locales "Ejemplo: Juegos".
- ***Sincronización Externa*** → Consiste en sincronizar / ordenar eventos "Ejemplo: Eventos en bolsa, GPS,..".

3.1 Métodos de Sincronización

- *UTC: Universal Coordinated Time*

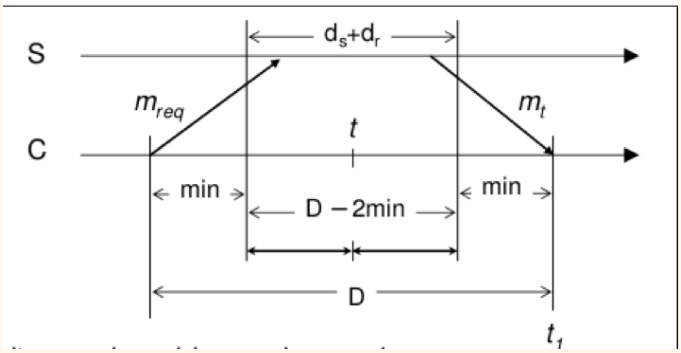
  Consiste en la transmisión de señal desde centros terrestres o satelitales, una o más máquinas del SD son receptoras de señal UTC.

- *Método Cristian*

  Es válido tanto para la sincronización interna como externa.

  Consiste en la utilización de un servidor de tiempo, conectado a un dispositivo que recibe señales de una fuente *UTC*, para poder así sincronizar computadores externos.

  Un proceso p solicita el tiempo en un mensaje "mreq" y recibe el valor del tiempo "t" en un mensaje "mt".



  Se puede medir este tiempo con precisión razonable si su ritmo de deriva de reloj es pequeño.

  Por tanto a menero "D" mejor precisión porque el tiempo de respuesta del servidor se decrementa. Sea:

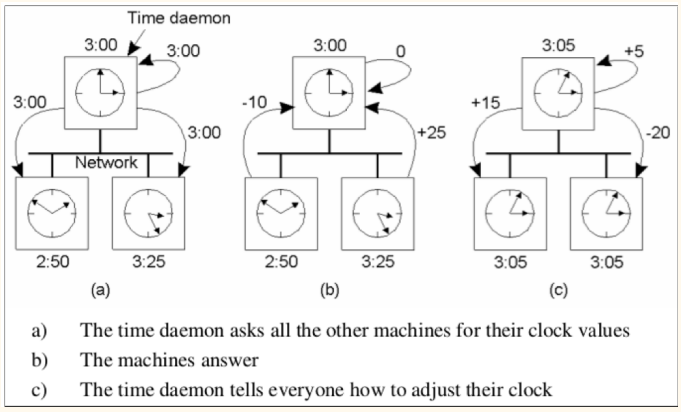
  - Desviación "D" = t - mt = t1 - D/2 - mt
  - Precisión = D/2 - min.

  El problema del Método Cristian es que es muy vulnerable a ataques de Denegación de Servicio "DDOS", que el servidor tenga un reloj que falla...

  Para solucionar estos problemas aparece el ***Algoritmo de Berkeley***.

- *Algoritmo de Berkeley*

  Basado en el Método de Cristian. En este algoritmo se selecciona un computador coordinador que actuará como "maestro".



a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock

A diferencia del Método Cristian, este computador consulta periódicamente a los otros computadores cuyos relojes están para ser sincronizados, llamados "esclavos".

Cada reloj "esclavo" envía su desviación respecto al "maestro". Para ajustar el sistema, se toma la hora media. Contra más dispositivos, mayor será la precisión.

A diferencia del Método de Cristian este es un método centralizado no distribuido, por tanto, solo sirve exclusivamente para la sincronización interna.

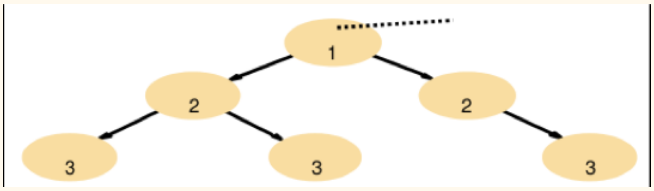
El nodo que difiere mucho del tiempo del maestro, no se tiene en cuenta a la hora de obtener el tiempo medio.

- *NTP: Network Time Protocol*

  Define una arquitectura para un servicio de tiempo y un protocolo para distribuir la información del tiempo.

  Es el estándar en internet, que proporciona una sincronización redundante con UTC.

  Está estructurado en capas "stratas" → Jerarquía:



Servidores primarios: Referencias a UTC fiables.

Servidores secundarios: Sincronizados con los primarios.

Los que están en la capa 1 son los relojes atómicos (hora casi perfecta), en la capa 2 encontramos servidores sincronizados con los de la capa 1. La sincronización es mejor contra más arriba se encuentre el servidor. Debido a los errores que se acumulan solamente se llega a la capa 3-5.

Tiene varios modos de operación:

- Red local son soporte adecuado: modo Multicast.
- Mayor precisión: modo de llamada a procedimiento.

- Mejor sincronización interna: Modo simétrico.

4. Tiempo Lógico

Surge debido a la dificultad de sincronizar el tiempo físico, como solo se quería saber que evento/mensaje ocurrió antes, apareció el tiempo lógico.

A cada proceso se le asigna un reloj lógico "contador". Podemos ordenar los procesos de una misma máquina. Si los eventos no suceden en el mismo proceso NO podemos ordenarlos.

La relación de causalidad " →", entre dos eventos nos dice que el evento "a" sucederá antes que el "b": " a → b".

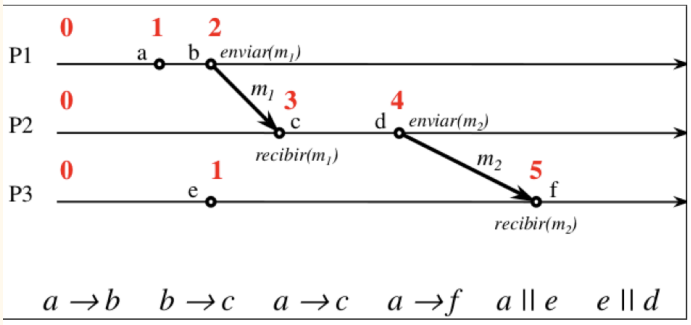
Si entre dos eventos no hay *relación de causalidad*, se dicen que son **concurrentes - paralelos "||"**: evento x || evento y.

4.1 Tiempo lógico de Lamport

Fue propuesto en 1978, para indicar la relaciones de causalidad entre procesos.

Cada proceso "P" tiene un reloj local "C" para asociar marcas de tiempo a sus eventos.

Luego se comparar los contadores de los procesos, se queda con el mayor y se incrementa en "1" el reloj "C".



Encontramos un problema: Si C(x) < C(y) no implica que "x → y"[2].

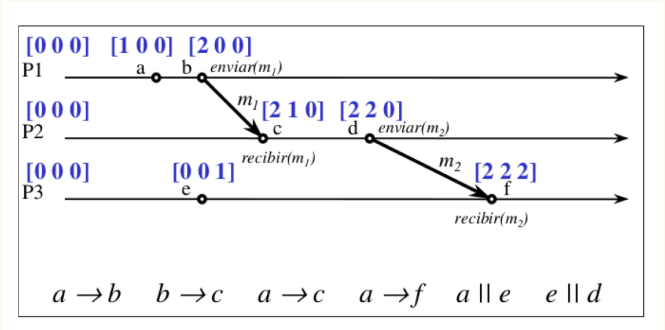
Este problema se soluciona mediante el Algoritmo de Vectores de Tiempo.

4.1.2 Algoritmo de Vectores de Tiempo

Encontramos relojes vectoriales, algoritmo más potente que Lamport.

Es un sistema de comunicación entre muchos nodos "Amazon".

Cuando un proceso envía un mensaje envía todos los contadores que conoce.



[2] Encontramos una relación de causalidad entre x e y "*x sucede antes que y*".

Ahora sí podemos utilizar las etiquetas para comparar eventos de diferentes procesos.

3 procesos → Vector con 3 contadores.

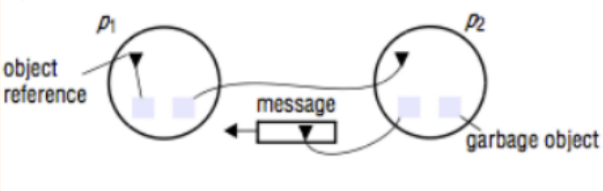
Si "Vx < Vy"[3], podemos decir que Vx se ha producido antes que Vy.

Si no podemos decir si los vectores son menor o iguales que otros, estos son concurrentes o paralelos → Vx || Vy.

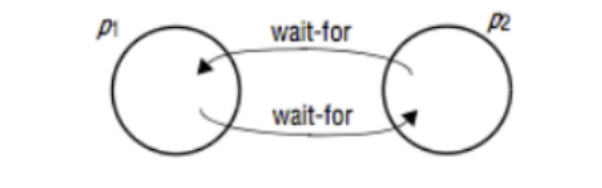
5. Estados globales

Hay tareas para las que necesitamos conocer el estado global del sistema.

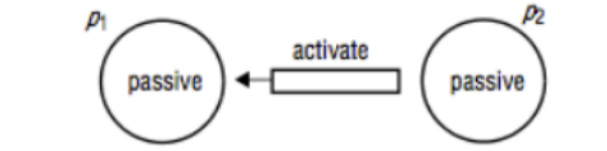
- ***Recolección de basura***: Un objeto se considera desechable si no hay posteriormente ninguna referencia a él desde cualquier parte del SD. La memoria ocupada por el objeto puede ser reclamada una vez que este se sabe que es desechable.



- ***Detección de interbloqueos***: Un bloqueo indefinido distribuido ocurre cuando cada uno de los procesos espera a otro proceso para enviarle un mensaje.



- ***Detección de estados de determinación***: Detectar que un algoritmo distribuido ha terminado.



Es vital tener en cuenta el estado de los procesos y del canal de comunicación.

[3] Vx y Vy, son relojes vectoriales, "x , y" son los procesos.

5.1 Cortes Consistentes

Un corte "C" es consistente si, para cada suceso que contienen también contiene todos los sucesos que "sucedieron antes que".

Estado global consistente es aquel que corresponde con un corte consistente.

Basándonos en los Vectores de Tiempo podemos saber si un corte es consistente o no:

$$\forall i, j : Vi[i](ei \hat{} ci) \geqslant Vj[i](ej \hat{} cj)$$

Puesto que cada proceso posee una visión parcial del sistema, para construir un corte consistente los procesos deben ejecutar un algoritmo establecido.

Utilidad: Detección de interbloqueos, establecimiento de puntos de recuperación de un sistema, finalización distribuida.

5.1.2 Monitorización

Depurar un SD requiere su estado global, para poder hacer evaluaciones de predicados en dichos estados, generalmente, la evaluación trata de determinar si el predicado cumple con la condición "posiblemente" o "sin duda alguna".

El proceso monitor solo registra los estados globales consistentes, pues son lo únicos en que podemos evaluar el predicado con certeza.

Monitorización del estado global:

- ***Distribuido***: Algoritmo de instantánea de Chandy y Lamport.
- ***Centralizado***: Algoritmo de Marzullo y Neiger:
    - Los procesos envían su estado inicial al proceso monitor periódicamente, le vuelven a enviar su estado.
    - El monitor registra los mensajes de estado en colas de proceso "Una por proceso".

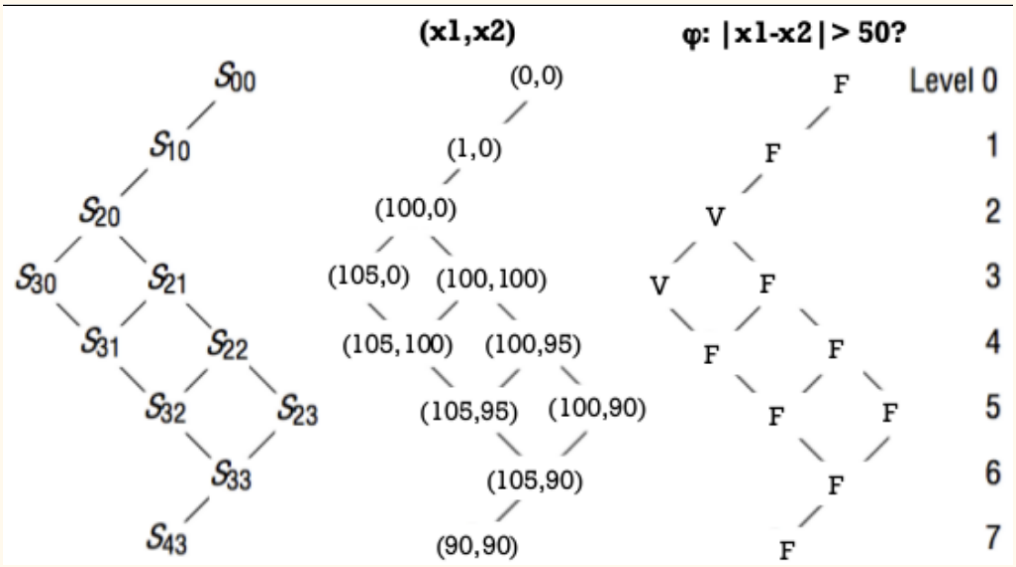
Podemos reducir el tráfico de mensajes de estado:

- ***Tamaño***: El predicado puede depender sólo de ciertas partes de un proceso "no es necesario mandar el estado completo".
- ***Número***: El cambio de valor del predicado solo ocurre en algunos casos "solo hay que recoger los estados en cambios relevantes".

5.1.3 Red de Estados Globales

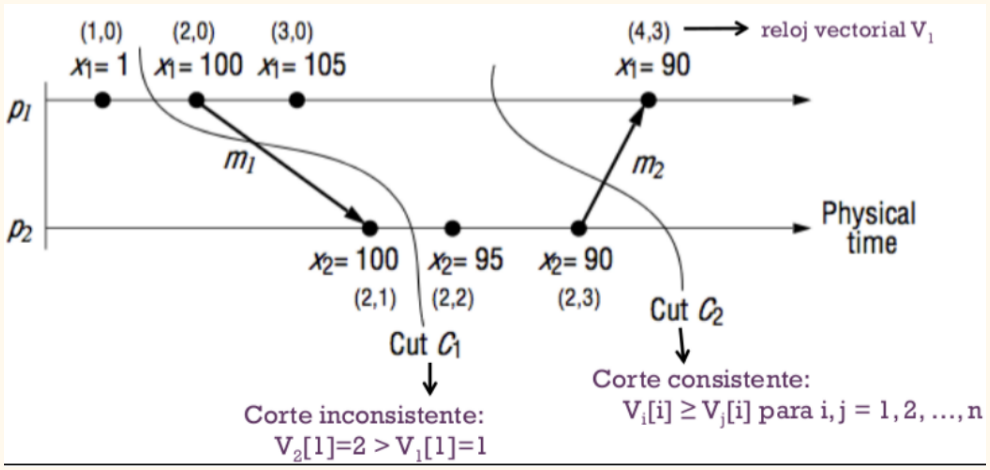
Encontramos 3 estados:

- ***Linealización***: Ruta entre estados.
- ***Posiblemente fi***: Existe un estado consistente "S" a través del que pasa una linealización tal que *fi(S)* = Verdadero.
- ***Sin ninguna alguna fi***: Existe un conjunto de estados consistentes "S*" a través del que pasan todas las linealizaciones, tal que, $\forall$ *S , S* , fi(S)* = Verdadero.



5.1.4 Algoritmo de instantánea de Chandy y Lamport

Objetivo: Obtener un conjunto de estados de procesos y del canal de comunicación (instantánea) que sea un estado global consistente.



Suposiciones:

- Los canales y procesos no fallan, todos los mensajes se reciben correctamente.
- Los canales son unidireccionales “FIFO”.
- Hay canales de comunicación directos entre todos los tipos.
- Cualquier proceso puede tomar una instantánea en cualquier momento.
- Los procesos pueden continuar su ejecución y comunicación mientras se está tomando una instantánea.

El algoritmo se define mediante dos reglas, la regla de recepción del marcador y la regla de envío del marcador.

La regla del envío del marcador obliga a los procesos a enviar un marcador después de haber registrado su estado, pero antes de que envíen cualquier otro mensaje.

En resumen, la Depuración Distribuida consiste en:

- Determinar el predicado que queremos evaluar.
- Especificar un método para construir una red o historia de estados globales consistentes “teniendo en cuenta el predicado para optimizar tráfico”.
- Evaluar si nuestro predicado se cumple en algún momento:
    - ***Si es posible***, se cumplirá para alguna linealización.
    - ***Si es sin duda***, se cumplirá para todas las linealizaciones.

Tema 4 Redes Peer-to-Peer

1. Introducción

Peer-to-Peer: Sistema auto-organizado de entidades iguales y autónomas "peers" cuyo objetivo es el uso compartido de recursos distribuidos en un ambiente de red evitando servicios centralizados.

¿Qué no es?: Red cliente-servidor

Frente a las redes P2P[4], se encuentran las que se basan en una arquitectura cliente-servidor, donde los clientes solicitan recursos a unos/varios servidores. En esta arquitectura, según aumenta el número de usuarios, la tasa de transferencia disminuye, debido a que los recursos de los que dispone el servidor se consumen debido al tráfico que genera.

Definiciones

- ***Descentralización*** → Los diferentes miembros "iguales" del sistema son propietarios y tienen el control de los datos y recursos de su ordenador, complicando la implementación de los sistemas P2P, ya que no hay una visión global del sistema.

  Por tanto, algunos sistemas P2P optan por sistemas híbridos "Napster o eDonkey" → directorios centralizados de ficheros descargados por nodos.

- ***Escalabilidad*** → Es una consecuencia directa de la Descentralización. El hecho de que las operaciones se hagan entre los peers hace que el sistema pueda soportar muchas más operaciones, ya que no tenemos nodos centralizados.
- ***Autoorganización*** → Es necesaria ya que acostumbran a ser sistemas de gran escala.
- ***Coste de la propiedad*** → En los sistemas P2P, la propiedad es compartida " los recursos que forman parte del sistema los aportan los propios participantes en el sistema". Esto reduce el coste de poseer el sistema, los contenidos y el mantenimineto.
- ***Conectividad puntual "ad-hoc connectivity"*** → Los peers solo se conectan para realizar actividades concretas y se vuelven a desconectar.

[4] P2P → Peer-to-Peer

- ***Rendimiento*** → Se mejora el rendimiento agregando capacidad de almacenamiento distribuido o capacidad de procesamiento en dispositivos repartidos por toda la red.
    - Replicación.
    - Uso de memorias cachés.
    - Encaminamiento inteligente.
- ***Seguridad*** → Tienen los mismos problemas que cualquier SD:
    - Cifrado multiclave.
    - Seguridad del código "que no ejecute o modifique cosas en el ordenador".
    - Gestión de derechos digitales "propiedad intelectual".
    - Reputación y contabilización "Como de bueno es un peer, se realiza contabilización de los recursos compartidos".
- ***Transparencia y usabilidad*** → Los sistemas P2P pueden funcionar independientemente del dispositivo de los peers "ordenadores, agendas electrónicas, teléfono móvil).

  Otra transparencia que tienen es proporcionar movilidad y seguridad.

- ***Resistencia a fallos*** → Uno de los objetivos de los sistemas P2P es evitar que si falla un componente del sistema, deje de funcionar todo el sistema. Por tanto, es deseable que haya particiones en la red para que los peers puedan continuar colaborando.

Ataques

Las redes P2P promueven la libre compartición de archivos y servicios, lo que lo hace muy vulnerable a determinados ataques:

- ***Poisoning "envenenamiento"***: Consiste en distribuir información y/o archivos falsos "que no se corresponden con la descripción dada".
- ***Pollution "contaminación"***: Insertar paquetes malignos dentro de legítimos, no sólo para afectar al rendimiento si no que a veces contienen malware[5].
- ***Ataques DoS, choking***: El tráfico de la red P2P puede ser excesivo, provocando que la red se ahogue. Algunos ataques pueden estar organizados para producir Denegación de Servicio.
- ***Identificación***: Algunas redes P2P permiten identificar a los usuarios.
- ***Scamming***: Muchas "empresas" se aprovechan de la ignorancia de los usuarios y le venden servicios especiales, que no es más que solo un acceso a lo que es gratis pero cobrándose.

[5] Malware → Software malicioso.

Almacenamiento

- *Manejo de Documentos*

  Encontramos problemas:

  - Organizados de forma distribuida “usualmente”.
  - Grandes porciones de documentos creados en una compañía son distribuidos entre PCs sin un repositorio central.

  Soluciones a estos problemas:

  - Las redes P2P crean un repositorio conectado “datos locales de los peers”.
  - Indexación y categorización de datos por cada peer.
  - Información autoorganizada desde áreas del conocimiento.

- *Archivos compartidos en P2P*

  Almacena contenidos en nodos individuales en vez de un lugar central.

  Peers que bajan archivos los ponen disponibles para otros peers.

  El 70% del tráfico de internet puede ser atribuido a intercambio de archivos.

- *Problemas de Búsqueda*

  Localizar los recursos es un problema central de las redes P2P al igual que compartir archivos en particular.

Procesamiento

- *Aumento de ciclos de procesador*

  Encontramos un incremento de Requerimientos para la Computación de Alto Rendimiento “bioinformática, logística,sector financiero”.

  Podemos computar entidades en redes con poco uso.

  Hacemos uso de aplicaciones P2P para uso de ciclos de procesador:

  - Formar clusters de computadoras independientes conectadas a la red, esto hace que para una computadora sea transparente y todos los nodos en red son combinados en una computadora lógica simple.
  - Grid computing
  - Logran un poder de computación equiparable a las supercomputadoras más potentes.

Arquitectura de las redes P2P

Existen tres modelos diferentes a la hora de crear una red P2P → directorio centralizado, descentralizado o puro, híbrido.

Siendo directorio centralizado o descentralizado "puro" → Primera generación de las redes P2P e Híbrido → Segunda generación de las redes P2P.

En la tercera generación, en las redes P2P se utiliza un middleware que se encarga de gestionar la aplicación independientemente de la distribución de los recursos a gran escala.

Con todo esto, se evita tener que implementar en la aplicación toda la gestión de los recursos.

Algunos Middlewares → Pastry, CAN y Tapestry.

- *Modelo centralizado*

  El servicio indexado es provisto centralmente por una entidad de coordinación. De esta forma, cuando un nodo quiere buscar un fichero dentro del sistema, este le hace un request al nodo coordinador y este le responde con el nodo donde se encuentra el fichero.

  Por tanto, se produce la descarga del fichero entre ambos nodos de forma ajena al servidor. Una vez descargado, los nodos notifican de esto.

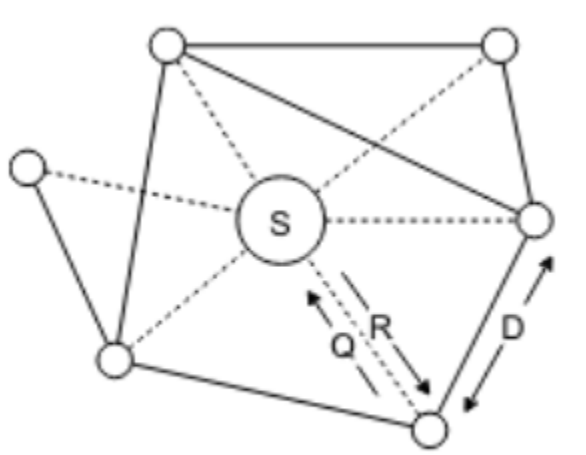


Imagen 2. Modelo centralizado

- *Modelo descentralizado "puro"*

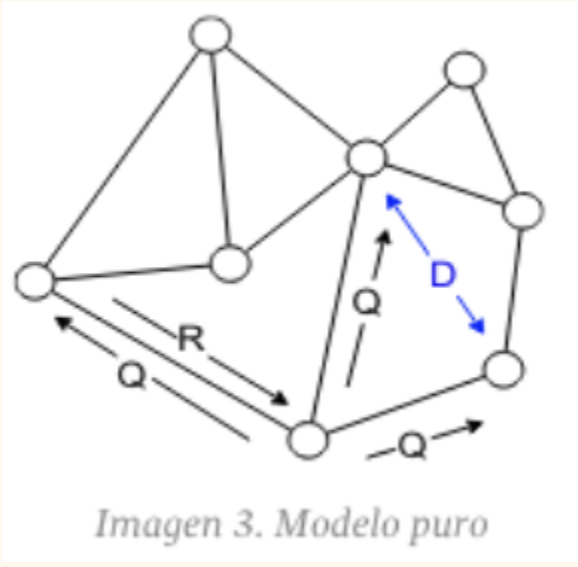


Imagen 3. Modelo puro

A diferencia del centralizado, no existe una autoridad central de coordinación, por tanto, todos los pares actúan por igual.

Cuando un nodo quiere un determinado fichero, hace un request a los peers con el que está conectado. Si el fichero, no se encuentra en esos nodos, pasa a los nodos que están conectados a estos, así sucesivamente hasta que encuentra el fichero.

Cuando el fichero es encontrado, se le envía la respuesta afirmativa al nodo inicial y comienza la descarga del fichero desde el o los nodos que lo ofrecen.

- *Modelo Híbrido*

  Aquí encontramos una capa de jerarquía. Esta capa es dinámica y está compuesta por varios servidores que no almacenan ningún recurso, sino que encaminan el tráfico y administra los recursos, sin conocer la identidad de cada nodo para no perder confiabilidad.

  Los servidores intercambian información entre sí.

Tipos de Estructuras

Las redes P2P pueden clasificarse por tener o no una estructura determinada en base a cómo se enlazan unos nodos a otros.

Encontramos:

- *No estructurada*

  La localización de recursos no está determinada en nodos concretos, sino que a cada uno se le asigna enlaces arbitrariamente, que se irán actualizando.

  Usan un mecanismo de búsqueda → No determinista, no garantiza que se vaya a encontrar el recursos aunque este esté en la red. Se debe a que los request llevan un limitador de profundidad.

- Estructuradas

  Son aquellas redes en las que los recursos están situados en nodos precisos. Cada uno de los nodos que forman la red cuenta con su propia tabla hash, cada usuario es responsable de una parte específica del contenido de la red.

  A los ficheros del sistema se les asignará una clave mediante una función hash.

  Una función hash es aquella que dado un objeto elabora una clave para él. Si dos elementos tienen la misma clave a partir de una misma función hash, decimos que el hash tiene colisiones. “No tiene inversa”.

  Cuando un usuario se descarga un fichero, si dispone de la clave hash, puede aplicarle la función para comprobar que el fichero es el mismo que el que está asociado a la clave “por tanto no se ha modificado”.

Jose Luis Venega Sánchez