Capítulo 4
Aplicar una blockchain: práctica

1. Arquitectura

1.1 Los sistemas distribuidos y consensos

1.1.1 ¿Qué es un sistema distribuido?

En el capítulo Introducción, hemos destacado las ventajas asociadas a un sistema distribuido. Como recordatorio, el término "sistema distribuido" significa una interconexión de una conjunto de ordenadores autónomos, de procesos o de procesadores. Los ordenadores, los procesos o los procesadores se llaman nudos. Para poder ser considerado autónomo, el nudo debe estar equipado, como mínimo, con su propio sistema de control. Para poder ser considerados interconectados, los nudos deben ser capaces de intercambiar información.

La definición también incluye un sistema de software construido como un conjunto de procesos comunicantes, incluso si funcionan en una instalación única. En la mayoría de los casos, un sistema distribuido está, como mínimo, formado por varios procesadores interconectados mediante una herramienta de comunicación.

En los artículos sobre el tema aparecen otras definiciones más restrictivas. Por ejemplo, un sistema es distribuido si y solo si la existencia de nudos autónomos es transparente para los usuarios del sistema.

Un sistema distribuido de esta manera se comporta como un sistema de ordenadores virtuales y stand-alone (independiente). Sin embargo, la implementación de esta transparencia necesita el desarrollo de algoritmos distribuidos de control complejo.

1.1.2 Red informática o red de ordenadores

Una red informática es un conjunto de ordenadores conectados por mecanismos de comunicación que permiten el intercambio de información. Esta comunicación se hace mediante el envío y la recepción de mensajes. En función de la distancia que separa los distintos ordenadores, se pueden distinguir dos redes: una red local *local area network* y una red extendida *wide area network*.

Habitualmente, una red extendida conecta organizaciones entre ellas (industriales, universitarias, etc.), separadas por una distancia física superior a unos 10 kilómetros. Cada nudo de la red presenta una instalación completa que incluye todos los periféricos y todos los programas. En cuanto a la red local, está compuesta de nudos en una misma organización.

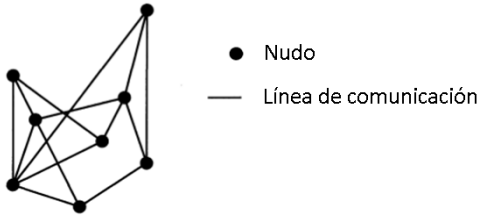
No es fácil establecer la diferencia entre las dos redes y, desde el punto de vista algorítmico, las dos son similares, porque los problemas algorítmicos suceden en todos los nudos de una red. Sin embargo, hay diferencias relacionadas con el aspecto del desarrollo de los algoritmos.

– Parámetro de fiabilidad: para las redes extendidas, durante el diseño de los algoritmos distribuidos nunca se debe ignorar la posibilidad de fallo de la transmisión de los mensajes. Para las redes locales, a menudo se descuida este parámetro porque son más fiables.
– Tiempo de comunicación: el tiempo de transmisión de los mensajes en una red extendida es mayor que en una red local. En una red extendida, a menudo se descuida el tiempo de cálculo frente al tiempo de transmisión de la información.
– Homogeneidad: en una red local, observamos una cierta homogeneidad en los programas y los protocolos utilizados. En las redes extendidas observamos variedad de protocolos y de programas, lo que hace surgir la problemática de conversión entre los nudos y de compatibilidad entre los distintos estándares.

– Confianza mutua: en una red local, la confianza entre los nudos puede considerarse cómo implícita, pero no sucede lo mismo en una red extendida. Las redes extendidas requieren el desarrollo de algoritmos seguros y protegidos contra los nudos maliciosos.

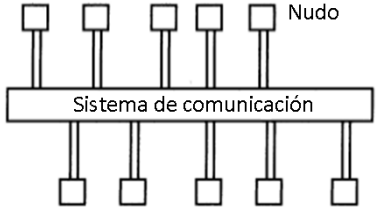
Tomando como base las características que acabamos de mencionar, podemos describir una red blockchain como un área extendida o local en función de las aplicaciones y de los nudos que la componen.

La comunicación entre los nudos se establece con ayuda de mecanismos como las líneas telefónicas, las fibras ópticas o bien las conexiones por satélite. Utilizando un vocabulario más técnico, la estructura de interconexión entre los nudos se representa mediante un grafo. La figura que aparece aquí debajo representa una red de conexión con forma de grafo.



Representación de una red extendida mediante un grafo

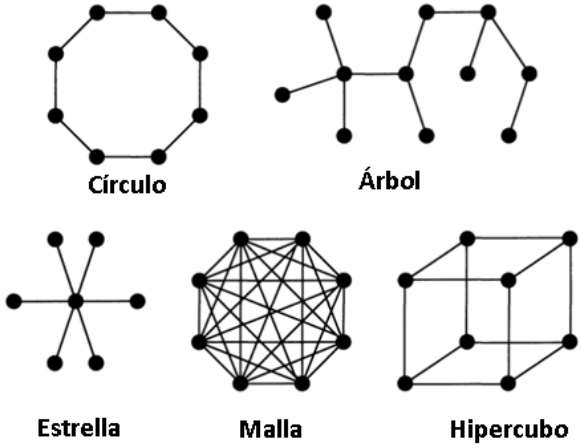
La comunicación entre los nudos se hace mediante un mecanismo único al que están conectado todos ellos. Este tipo de estructura se llama estructura en bus. En la figura siguiente podemos ver una representación esquemática.



Estructura en bus de una red local

La topología de una red

Después de esta introducción a la comunicación en una red distribuida con mensajes en tránsito, vamos a interesarnos por la topología de estas redes. De manera general, no es necesario que cada proceso tenga capacidad para enviar un mensaje al resto de procesos. En función del emisor y del destinatario de los mensajes, hay varias topologías de red posibles (ver figura más abajo). Sí un proceso p puede enviar un mensaje a un proceso q, se ha creado un canal de comunicación. Si el canal de comunicación también permite la comunicación inversa, desde q a p_1, se dice que es bidireccional; en caso contrario, es unidireccional.



Ejemplos de topologías de las redes más utilizadas

1.1.3 Problemas algorítmicos

El uso de una red local para la ejecución de una aplicación distribuida necesita una solución a los siguientes problemas de control distribuido.

- **Transmisión y sincronización**: si hay que poner la información a disposición de todos los procesadores o si todos los procesadores tienen que esperar hasta que se cumpla una condición global, es necesario que la información se transmita a todos los procesadores.
- **Elección**: algunas tareas deben realizarse mediante un único proceso, por ejemplo, la iniciación de una estructura de datos o la generación de salidas. Si no hay ningún proceso designado a priori para realizar esta tarea (incluso si a veces es necesario), se debe ejecutar un algoritmo distribuido para seleccionar el proceso que se ocupará de esta tarea.
- **Detección de terminaciones**: para los procesos dentro de un sistema distribuido, no siempre es posible saber si ha terminado el cálculo en el que están tomando parte. La detección de las terminaciones es necesaria para tener un resultado definitivo del cálculo en curso.
- **Asignación de recursos**: es posible que un nudo necesite acceder a ciertos recursos disponibles en la red, aunque el acceso a estos recursos no indique su ubicación. La elaboración de una tabla que permite inventariar la ubicación de los recursos disponibles puede resultar inadecuada según la cantidad de los mencionados recursos y de su migración a uno u otro nudo. Para resolver este problema, se han desarrollado algoritmos basados en ondas (Baratz, 1987).
- **Exclusión mutua**: este problema aparece cuando los procesos deben intervenir en un mismo recurso, que solo puede ser utilizado por un único proceso en un instante dado. Un ejemplo de este tipo de recursos es la impresora o la escritura en un archivo. Así, se necesitan algoritmos distribuidos para determinar, si varios procesos demandan un acceso simultáneo, cuál utilizará primero el recurso y garantizar que el proceso siguiente empieza cuando termina el anterior.

– **Detección de puntos muertos**: en caso de que los procesos deban esperar el final de los procesos anteriores (por ejemplo: cuando comparten un recurso), podemos ver que aparece una espera cíclica en la que no es posible hacer ningún cálculo. Hay que detectar estas situaciones de punto muerto y se deben tomar acciones para reiniciar los cálculos o proseguir con ellos.

1.1.4 Cooperación de procesos

El diseño de un sistema de software podría simplificarse mediante la organización del software en grupos de procesos secuenciales donde cada proceso tiene su propia tarea bien definida. Un ejemplo para ilustrar esta simplificación es la conversión de Conway. El problema consiste en leer 80 caracteres y en reescribir los mismos datos en 125 caracteres. Luego, se trata de añadir espacios en blanco después de cada carácter y de cambiar cada par de asteriscos ("**") por una exclamación de cierre ("!"). La conversión puede dirigirla un solo programa, pero escribirlo puede resultar muy complicado. El programa se puede estructurar mejor con dos procesos cooperativos. El primer proceso p_1 lee los datos en la entrada y los convierte en caracteres imprimibles. El segundo proceso p_2 recibe los caracteres y procede a la inserción de los caracteres en blanco así como al cambio por exclamaciones de cierre.

Este tipo de arquitectura impone que la aplicación sea distribuida de manera lógica.

Algunos procesos se ejecutan en el mismo procesador y comparten la memoria física. En este contexto, el sistema distribuido se puede enfrentar a algunos problemas como los que se describen a continuación

– **Atomicidad de la memoria de las operaciones**: con frecuencia se admite que la lectura y la escritura en una palabra de memoria son atómicas. Dicho de otro modo, la lectura o la escritura ejecutada por un procesador termina antes de que se inicie otra operación de lectura o de escritura. Si se utilizan estructuras más largas que una palabra, las operaciones tienen que sincronizarse para evitar una lectura parcial de la estructura actualizada. Esta sincronización se puede hacer mediante la implementación de una exclusión mutua en la estructura: si un proceso tiene acceso a la estructura, ningún otro proceso puede tener acceso a ella (Dijkstra, 1968). La condición de exclusión mutua puede inducir una reducción del rendimiento del proceso.

– **Problema de los productores consumidores**: dos procesos, uno de escritura en una consola y otro de lectura en la misma consola, deben coordinarse y sincronizarse para evitar escribir en una consola llena o leer en una consola vacía.
– **Recolección de basura o "garbage collection"**: una aplicación programada utilizando una estructura de datos dinámica puede producir celdas de memoria inaccesibles que se llaman "basura". Formalmente, cuando del sistema de memoria es insuficiente hay que interrumpir una aplicación, para permitir a un programa llamado recolector de basura que identifique y recoja la memoria inaccesible (Dijkstra, 1978). Es indispensable que haya una cooperación estrecha entre la aplicación y el recogedor, porque la aplicación tiene que modificar la estructura de los punteros en la memoria cuando el recogedor decide qué celda es inaccesible.

Las soluciones a estos problemas de interacción de los procesos en un procesador son extremadamente sofisticadas y, a veces, la ligera superposición de las etapas de los distintos procesos puede ser el origen de errores en resultados que parecen correctos a primera vista. Por lo tanto, los sistemas de operación y los lenguajes de programación ofrecen soluciones primitivas y una organización estructurada para la comunicación entre procesos.

– **Los semáforos**: un semáforo es una variable o un entero no negativo cuyo valor puede ser leído o escrito en una única operación atómica. Una operación V incrementa su valor y una operación P lo decrementa (el proceso se suspende cuando su valor es igual a 0). Los semáforos son herramientas apropiadas para implementar la exclusión mutua en una estructura de datos compartidos.
– **Los monitores**: un monitor es una estructura de datos y un conjunto de procedimientos que se pueden ejecutar con los datos llamando a los procesos de una manera mutuamente exclusiva. Esto permite garantizar un uso correcto de los datos.
– **Los pipes**: un pipe es un mecanismo de transferencia de flujo de datos de un proceso a otro sincronizándolos. Es una solución preprogramada para el problema productor-consumidor.