2. Conceptos de los sistemas distribuidos

Los sistemas distribuidos están basados en las ideas básicas de transparencia, eficiencia, flexibilidad, escalabilidad y fiabilidad. Sin embargo estos aspectos son en parte contrarios, y por lo tanto los sistemas distribuidos han de cumplir en su diseño el compromiso de que todos los puntos anteriores sean solucionados de manera aceptable.

Transparencia

El concepto de transparencia de un sistema distribuido va ligado a la idea de que todo el sistema funcione de forma similar en todos los puntos de la red, independientemente de la posición del usuario. Queda como labor del sistema operativo el establecer los mecanismos que oculten la naturaleza distribuida del sistema y que permitan trabajar a los usuarios como si de un único equipo se tratara.

En un sistema transparente, las diferentes copias de un archivo deben aparecer al usuario como un único archivo. Queda como labor del sistema operativo el controlar las copias, actualizarlas en caso de modificación y en general, la unicidad de los recursos y el control de la concurrencia.

El que el sistema disponga de varios procesadores debe lograr un mayor rendimiento del sistema, pero el sistema operativo debe controlar que tanto el usuario como los programadores vean el núcleo del sistema distribuido como un único procesador. El paralelismo es otro punto clave que debe controlar el sistema operativo, que debe distribuir las tareas entre los distintos procesadores como en un sistema multiprocesador, pero con la dificultad añadida de que ésta tarea hay que realizarla a través de varios ordenadores.

Eficiencia

La idea base de los sistemas distribuidos es la de obtener sistemas mucho más rápidos que los ordenadores actuales. Es en este punto cuando nos encontramos de nuevo con el paralelismo.

Para lograr un sistema eficiente hay que descartar la idea de ejecutar un programa en un único procesador de todo el sistema, y pensar en distribuir las tareas a los procesadores libres más rápidos en cada momento.

La idea de que un procesador vaya a realizar una tarea de forma rápida es bastante compleja, y depende de muchos aspectos concretos, como la propia velocidad del procesador, pero también la localidad del procesador, los datos, los dispositivos, etc. Se han de evitar situaciones como enviar un trabajo de impresión a un ordenador que no tenga conectada una impresora de forma local.

Flexibilidad

Un proyecto en desarrollo como el diseño de un sistema operativo distribuido debe estar abierto a cambios y actualizaciones que mejoren el funcionamiento del sistema. Esta necesidad ha provocado una diferenciación entre las dos diferentes arquitecturas del núcleo del sistema operativo: el núcleo monolítico y el micro núcleo. Las diferencias entre

ambos son los servicios que ofrece el núcleo del sistema operativo. Mientras el núcleo monolítico ofrece todas las funciones básicas del sistema integradas en el núcleo, el micro núcleo incorpora solamente las fundamentales, que incluyen únicamente el control de los procesos y la comunicación entre ellos y la memoria. El resto de servicios se cargan dinámicamente a partir de servidores en el nivel de usuario.

Núcleo monolítico

Como ejemplo de sistema operativo de núcleo monolítico está UNIX. Estos sistemas tienen un núcleo grande y complejo, que engloba todos los servicios del sistema. Está programado de forma no modular, y tiene un rendimiento mayor que un micro núcleo. Sin embargo, cualquier cambio a realizar en cualquier servicio requiere la parada de todo el sistema y la recopilación del núcleo.

Micro núcleo

La arquitectura de micro núcleo ofrece la alternativa al núcleo monolítico. Se basa en una programación altamente modular, y tiene un tamaño mucho menor que el núcleo monolítico. Como consecuencia, el refinamiento y el control de errores son más rápidos y sencillos. Además, la actualización de los servicios es más sencilla y ágil, ya que sólo es necesaria la recopilación del servicio y no de todo el núcleo. Como contraprestación, el rendimiento se ve afectado negativamente.

En la actualidad la mayoría de sistemas operativos distribuidos en desarrollo tienden a un diseño de micro núcleo. Los núcleos tienden a contener menos errores y a ser más fáciles de implementar y de corregir. El sistema pierde ligeramente en rendimiento, pero a cambio consigue un gran aumento de la flexibilidad.

Escalabilidad

Un sistema operativo distribuido debería funcionar tanto para una docena de ordenadores como varios millares. Igualmente, debería no ser determinante el tipo de red utilizada (LAN o WAN) ni las distancias entre los equipos, etc.

Aunque este punto sería muy deseable, puede que las soluciones válidas para unos cuantos ordenadores no sean aplicables para varios miles. Del mismo modo el tipo de red condiciona tremendamente el rendimiento del sistema, y puede que lo que funcione para un tipo de red, para otro requiera un nuevo diseño.

La escalabilidad propone que cualquier ordenador individual ha de ser capaz de trabajar independientemente como un sistema distribuido, pero también debe poder hacerlo conectado a muchas otras máquinas.

Fiabilidad

Una de las ventajas claras que nos ofrece la idea de sistema distribuido es que el funcionamiento de todo el sistema no debe estar ligado a ciertas máquinas de la red, sino que cualquier equipo pueda suplir a otro en caso de que uno se estropee o falle.

La forma más evidente de lograr la fiabilidad de todo el sistema está en la redundancia. La información no debe estar almacenada en un solo servidor de archivos, sino en por lo menos dos máquinas. Mediante la redundancia de los principales archivos o de todos

evitamos el caso de que el fallo de un servidor bloquee todo el sistema, al tener una copia idéntica de los archivos en otro equipo.

Otro tipo de redundancia más compleja se refiere a los procesos. Las tareas críticas podrían enviarse a varios procesadores independientes, de forma que el primer procesador realizaría la tarea normalmente, pero ésta pasaría a ejecutarse en otro procesador si el primero hubiera fallado.

Comunicación

La comunicación entre procesos en sistemas con un único procesador se lleva a cabo mediante el uso de memoria compartida entre los procesos. En los sistemas distribuidos, al no haber conexión física entre las distintas memorias de los equipos, la comunicación se realiza mediante la transferencia de mensajes.

3. El estándar ISO OSI

Para el envío de mensajes se usa el estándar ISO OSI (interconexión de sistemas abiertos), un modelo por capas para la comunicación de sistemas abiertos. Las capas proporcionan varias interfaces con diferentes niveles de detalle, siendo la última la más general. El estándar OSI define las siguientes siete capas: física, enlace de datos, red, transporte, sesión, presentación y aplicación.

El modelo OSI distingue dos tipos de protocolos, los orientados hacia las conexiones y los protocolos sin conexión. En los primeros, antes de cualquier envío de datos se requiere una conexión virtual, que tras el envío deben finalizar. Los protocolos sin conexión no requieren este paso previo, y los mensajes se envían en forma de datagramas.

4. Modo de transmisión asíncrona ATM

El modo de transmisión asíncrona o ATM proporciona un rápido modo de transmisión. Las altas velocidades se alcanzan prescindiendo de la información de control de flujo y de control de errores en los nodos intermedios de la transmisión. ATM usa el modo orientado a conexión, y permite la transmisión de diferentes tipos de información, como voz, vídeo, datos, etc.

El modelo cliente-servidor basa la comunicación en una simplificación del modelo OSI. Las siete capas que proporciona producen un desaprovechamiento de la velocidad de transferencia de la red, con lo que sólo se usarán tres capas: física (1), enlace de datos (2) y solicitud/respuesta (5). Las transferencias se basan en el protocolo solicitud/respuesta y se elimina la necesidad de conexión.

RPC

Otro paso en el diseño de un sistema operativo distribuido plantea las llamadas a procedimientos remotos o RPCs. Los RPC amplían la llamada local a procedimientos, y los generalizan a una llamada a un procedimiento localizado en cualquier lugar de todo el sistema distribuido. En un sistema distribuido no se debería distinguir entre llamadas locales y RPCs, lo que favorece en gran medida la transparencia del sistema.

Una de las dificultades más evidentes a las que se enfrenta el RPC es el formato de los parámetros de los procedimientos. Un ejemplo para ilustrar este problema es la

posibilidad de que en un sistema distribuido formado por diferentes tipos de ordenadores, un ordenador con formato little endian llamara a un procedimiento de otro ordenador con formato big endian, etc. Este problema se podría solucionar si tenemos en cuenta que ambos programas conocen el tipo de datos de los parámetros, o estableciendo un estándar en el formato de los parámetros, de forma que sea usado de forma única.

Otro problema de peor solución es el paso de apuntadores como parámetros. Debido a que los apuntadores guardan una dirección del espacio de direcciones local, el procedimiento que recibe el apuntador como parámetro no puede usar inmediatamente el apuntador, ya que no tiene acceso a los datos, que para él son remotos. En el tema 7 se describirá la memoria compartida, que propone una solución a este problema.

Por último queda por solucionar la tolerancia a fallos. Una llamada a un procedimiento remoto puede fallar por motivos que antes no existían, como la pérdida de mensajes o el fallo del cliente o del servidor durante la ejecución del procedimiento.

La limitación del RPC más clara en los sistemas distribuidos es que no permite enviar una solicitud y recibir respuesta de varias fuentes a la vez, sino que la comunicación se realiza únicamente entre dos procesos. Por motivos de tolerancia a fallos, bloqueos, u otros, sería interesante poder tratar la comunicación en grupo.

5. Comunicación en grupo

La comunicación en grupo tiene que permitir la definición de grupos, así como características propias de los grupos, como la distinción entre grupos abiertos o que permiten el acceso y cerrados que lo limitan, o como la distinción del tipo de jerarquía dentro del grupo. Igualmente, los grupos han de tener operaciones relacionadas con su manejo, como la creación o modificación.

Sincronización

La sincronización en sistemas de un único ordenador no requiere ninguna consideración en el diseño del sistema operativo, ya que existe un reloj único que proporciona de forma regular y precisa el tiempo en cada momento. Sin embargo, los sistemas distribuidos tienen un reloj por cada ordenador del sistema, con lo que es fundamental una coordinación entre todos los relojes para mostrar una hora única. Los osciladores de cada ordenador son ligeramente diferentes, y como consecuencia todos los relojes sufren un desfase y deben ser sincronizados continuamente. La sincronización no es trivial, porque se realiza a través de mensajes por la red, cuyo tiempo de envío puede ser variable y depender de muchos factores, como la distancia, la velocidad de transmisión o la propia saturación de la red, etc.

El reloj

La sincronización no tiene por qué ser exacta, y bastará con que sea aproximadamente igual en todos los ordenadores. Hay que tener en cuenta, eso sí, el modo de actualizar la hora de un reloj en particular. Es fundamental no retrasar nunca la hora, aunque el reloj adelante. En vez de eso, hay que ralentizar la actualización del reloj, frenarlo, hasta que alcance la hora aproximadamente. Existen diferentes algoritmos de actualización de la hora, tres de ellos se exponen brevemente a continuación.

Algoritmo de Lamport

Tras el intento de sincronizar todos los relojes, surge la idea de que no es necesario que todos los relojes tengan la misma hora exacta, sino que simplemente mantengan una relación estable de forma que se mantenga la relación de qué suceso ocurrió antes que otro suceso cualquiera.

Este algoritmo se encarga exclusivamente de mantener el orden en que se suceden los procesos. En cada mensaje que se envía a otro ordenador se incluye la hora. Si el receptor del mensaje tiene una hora anterior a la indicada en el mensaje, utiliza la hora recibida incrementada en uno para actualizar su propia hora.

Algoritmo de Cristian

Consiste en disponer de un servidor de tiempo, que reciba la hora exacta. El servidor se encarga de enviar a cada ordenador la hora. Cada ordenador de destino sólo tiene que sumarle el tiempo de transporte del mensaje, que se puede calcular de forma aproximada.

Algoritmo de Berkeley

La principal desventaja del algoritmo de Cristian es que todo el sistema depende del servidor de tiempo, lo cual no es aceptable en un sistema distribuido fiable.

El algoritmo de Berkeley usa la hora de todos los ordenadores para elaborar una media, que se reenvía para que cada equipo actualice su propia hora ralentizando el reloj o adoptando la nueva hora, según el caso.

6. Otros problemas de sincronización

El reloj es únicamente uno de tantos problemas de sincronización que existen en los sistemas distribuidos. A continuación planteamos otros problemas relacionados con la sincronización.

En el momento de modificar unos datos compartidos, los procesos deben lograr la exclusión mutua que garantice que dos procesos no modifiquen los datos a la vez.

Algunos algoritmos distribuidos requieren que un proceso funcione como coordinador. Es necesario establecer ciertos algoritmos de elección de estos procesos.

Es necesario ocultar las técnicas de sincronización mediante la abstracción de las transacciones atómicas, que permitan a los programadores salvar los detalles de la programación con sincronización.

Soluciones frente a bloqueos son bastante más complejas que en sistemas con un único procesador.

7. Sistema de archivos

A diferencia de los sistemas de archivos clásicos, un sistema de archivos distribuido debe ser descentralizado, transparente y tolerante a fallos.

Transparencia

El problema más importante a resolver es el modo de que todos los ordenadores puedan acceder a todos los archivos del sistema. Para ello es necesario que todos los ordenadores lleven siempre y en todo momento una copia actualizada de la estructura de archivos y directorios. Si esta estructura oculta la localización física de los archivos entonces hemos cumplido el criterio de transparencia.

Fallos del sistema

Que el sistema de archivos sea tolerante a fallos implica que el sistema debe guardar varias copias del mismo archivo en distintos ordenadores para garantizar la disponibilidad en caso de fallo del servidor original. Además, se ha de aplicar un algoritmo que nos permita mantener todas las copias actualizadas de forma consistente, o un método alternativo que sólo nos permita acceder al archivo actualizado, como invalidar el resto de copias cuando en cualquiera de ellas se vaya a realizar una operación de escritura. El uso de memorias cache para agilizar el acceso a los archivos también es recomendable, pero este caso requiere analizar con especial atención la consistencia del sistema.

Modelos de acceso

Debido a la complejidad del acceso a los archivos a través de todo el sistema distribuido, surgen dos modelos para el acceso a los archivos: el modelo carga/descarga, y el modelo de acceso remoto. El primer modelo simplifica el acceso permitiendo únicamente las operaciones de cargar y descargar un archivo. El acceso a cualquier parte del archivo implica solicitar y guardar una copia local del archivo completo, y sólo se puede escribir de forma remota el archivo completo. Este método sería especialmente ineficaz a la hora de realizar pequeñas modificaciones en archivos muy grandes, como podrían ser bases de datos. El modelo de acceso remoto es mucho más complejo, y permite todas las operaciones típicas de un sistema de archivos local.

Memoria compartida distribuida

La memoria compartida distribuida o DSM es una abstracción que se propone como alternativa a la comunicación por mensajes.

Memoria compartida basada en páginas

El esquema de DSM propone un espacio de direcciones de memoria virtual que integre la memoria de todas las computadoras del sistema, y su uso mediante paginación. Las páginas quedan restringidas a estar necesariamente en un único ordenador. Cuando un programa intenta acceder a una posición virtual de memoria, se comprueba si esa página se encuentra de forma local. Si no se encuentra, se provoca un fallo de página, y el sistema operativo solicita la página al resto de computadoras. El sistema funciona de forma análoga al sistema de memoria virtual tradicional, pero en este caso los fallos de página se propagan al resto de ordenadores, hasta que la petición llega al ordenador que tiene la página virtual solicitada en su memoria local. A primera vista este sistema parece más eficiente que el acceso a la memoria virtual en disco, pero en la realidad ha mostrado ser un sistema demasiado lento en ciertas aplicaciones, ya que provoca un tráfico de páginas excesivo.

Una mejora dirigida a mejorar el rendimiento sugiere dividir el espacio de direcciones en una zona local y privada y una zona de memoria compartida, que se usará únicamente

por procesos que necesiten compartir datos. Esta abstracción se acerca a la idea de programación mediante la declaración explícita de datos públicos y privados, y minimiza el envío de información, ya que sólo se enviarán los datos que realmente vayan a compartirse.

Memoria compartida basada en objetos

Una alternativa al uso de páginas es tomar el objeto como base de la transferencia de memoria. Aunque el control de la memoria resulta más complejo, el resultado es al mismo tiempo modular y flexible, y la sincronización y el acceso se pueden integrar limpiamente. Otra de las restricciones de este modelo es que todos los accesos a los objetos compartidos han de realizarse mediante llamadas a los métodos de los objetos, con lo que no se admiten programas no modulares y se consideran incompatibles.

Modelos de consistencia

La duplicidad de los bloques compartidos aumenta el rendimiento, pero produce un problema de consistencia entre las diferentes copias de la página en caso de una escritura. Si con cada escritura es necesario actualizar todas las copias, el envío de las páginas por la red provoca que el tiempo de espera aumente demasiado, convirtiendo este método en impracticable. Para solucionar este problema se proponen diferentes modelos de consistencia, que establezcan un nivel aceptable de acercamiento tanto a la consistencia como al rendimiento. Nombramos algunos modelos de consistencia, del más fuerte al más débil: consistencia estricta, secuencial, causal, PRAM, del procesador, débil, de liberación y de entrada.

8. Bibliografía

Andrew S. Tanenbaum: "Sistemas Operativos Distribuidos", Prentice Hall (1996).

George Coulouris, Jean Dollimore, Tim Kindberg: "Distributed Systems, Concepts and Design", Addison-Wesley (1994).