Introducción

Ricardo Jiménez-Peris, Marta Patiño-Martínez

Distributed Systems Laboratory
Universidad Politécnica de Madrid (UPM)
http://lsd.ls.fi.upm.es/lsd/lsd.htm

La producción de este material ha sido financiado parcialmente for Microsoft Research Cambridge (Award MS-2004-393)

Temario

- Introducción a los sistemas distribuidos y tolerancia a fallos.
- Coordinación y acuerdo.
- Transacciones.
- Replicación.
- Tecnología. Casos de estudio.

Bibliografía

- Libros generales sobre sistemas distribuidos:
  - Distributed Systems: Concepts and Design. G. Colouris, J. Dollimore, T. Kindberg. 3rd edition, Addison-Wesley, 2000.
  - Distributed Systems: Principles and Paradigms. A. Tannenbaum & M. van Steen. Prentice Hall. 1995.
  - Distributed Systems, S. Mullender, ed. ACM-Press. 2nd Ed. Addison-Wesley, 1993.
  - Building Secure and Reliable Network Applications. K. Birman. Manning, 1996.
  - Distributed Systems for System Architects. P. Veríssimo, Luís Rodrigues. Kluwer, 2001.
  - Distributed Algorithms. N. Lynch. Morgan-Kaufmann, 1996.
  - Distributed Computing. H. Attiya and J. Welch. McGraw Hill. 1998.

Definición

- Un sistema distribuido es aquel cuyos componentes son ordenadores independientes conectados a través de una red.
- Características intrínsecas de un sistema distribuido:
  - Sus componentes computan de forma concurrente.
  - No existe un reloj global.
  - No comparten memoria.
  - Sus componentes fallan de forma independiente (idealmente).

Características

- Características deseables de los sistemas distribuidos:
  - Transparencia.
  - Escalabilidad.
  - Control de concurrencia.
  - Tolerancia a fallos.

Características: Transparencia

- La transparencia puede aplicarse a distintos conceptos:
  - Heterogeneidad: Los recursos se acceden del mismo modo independientemente de la arquitectura de los nodos, arquitectura de la red, sistema operativo, lenguaje de programación y fabricante del software (p.ej. Middleware como Corba o J2EE).
  - Acceso: Recursos locales y remotos se acceden del mismo modo (p. ej. Modelo de objetos remotos de Corba).
  - Ubicación: Los recursos pueden ser accedidos sin conocer su ubicación física (p. ej. Mediante servicios de nombrado o de descubrimiento de recursos).

Características: Transparencia

- Replicación: Un recurso replicado que puede accederse del mismo modo que uno no replicado.
- Fallos: Los recursos siguen pudiéndose acceder del mismo modo a pesar de que haya fallos.
- Movilidad: Los recursos pueden desplazarse sin afectar a su operación.
- Rendimiento: El sistema equilibra la carga *(load balancing)* sin que el acceso a los recursos se vea afectado.

Características: Escalabilidad

- Parámetros importantes:
  - Rendimiento (throughput): Número de operaciones por unidad de tiempo.
  - Tiempo de respuesta (response time): Tiempo desde que el cliente solicita un servicio hasta que recibe la respuesta.
  - Fiabilidad (reliability): Fiabilidad del sistema con respecto a la fiabilidad de los componentes.
- Un sistema distribuido es escalable si su capacidad de procesamiento puede crecer añadiendo nodos adicionales:
  - Aumenta el rendimiento con un número creciente de nodos (idealmente de forma lineal).
  - El tiempo de respuesta decrece (o se mantiene cte. o crece lentamente) con un número creciente de nodos.
  - La fiabilidad el sistema aumenta con número creciente de nodos (idealmente de forma logarítmica).

Características: Control de Concurrencia

- Los recursos deben poder accederse concurrentemente por distintos usuarios sin que pierdan su coherencia.
- Dos definiciones de coherencia:
  - Linearidad (*linearizability*): El resultado de llamadas concurrentes a un recurso debe ser equivalente a una ejecución secuencial de éstas.
  - Serialidad (*serializability*): El resultado de secuencias de operaciones (transacciones) ejecutadas concurrentemente por distintos usuarios deben ser equivalentes a la ejecución secuencial de cada una de las transacciones.

Características: Tolerancia a Fallos

- Los fallos en un sistema distribuido son intrínsecamente parciales.
- Los fallos parciales no deben afectar al sistema que debe seguir ofreciendo servicio, quizás en modo degradado (*graceful degradation*).
- Las dos propiedades más importantes son:
  - Disponibilidad *(availability):* Un recurso permanece disponible a pesar de que haya fallos.
  - Atomicidad *(atomicity):* Un recurso mantiene su consistencia a pesar de que haya fallos.

Características: Tolerancia a Fallos

- Niveles de tratamiento de fallos:
  - Detección. Ej.: Checksum.
  - Recuperación.
    - Hacia atrás.
    - Hacia adelante.
  - Enmascaramiento.
- Técnicas para tratar los fallos:
  - Redundancia:
    - temporal (p. ej., retransmisión de un mensaje),
    - espacial (replicación),
    - de diseño/valor (diversidad).

Modelos Arquitectónicos

- **Acceso remoto:** Un sistema central es accedido por un terminal a través de una red.
- **Modelo cliente-servidor:** La aplicación se ejecuta en el cliente mientras que los recursos se almacenan en servidores que son accedidos por los clientes a través de la red. Los clientes contienen interfaz de usuario y código de la aplicación.

Modelos Arquitectónicos

- **Arquitecturas en tres/cuatro capas (3/4-tier architectures):**
  - Los clientes tan sólo contienen software para la interacción con el usuario *(thin clients)*.
  - Las peticiones de los clientes son manejadas por un frontend (p.ej. un servidor web) que las redirige a las aplicaciones que las van a procesar.
  - Las aplicaciones residen en servidores de aplicación *(stateless o stateful)* y son invocadas por los clientes.
  - Los recursos residen en servidores de datos que son accedidos por las aplicaciones.

Modelos Arquitectónicos

- **Sistemas móviles:**
  - **Código móvil:** Los componentes están fijos, pero el código de las aplicaciones puede desplazarse (p. ej. agentes móviles).
  - **Nodos móviles:** Sistemas cuyos componentes pueden desplazarse por distintos puntos de la red: redes inalámbricas, ordenares portátiles, PDAs, etc. que se desconectan y se vuelven a conectar en otro punto de la red (p.ej. redes ad-hoc).

Modelos Arquitectónicos

- **Arquitecturas basadas en eventos:** La interacción es asíncrona, unos componentes producen mensajes y otros los consumen, pero no tienen por qué estar interconectados durante la interacción (p. ej.: JMS o Corba Event Service).
- **Sistemas *peer-to-peer*:** Sistemas en los que la interacción entre los componentes está totalmente descentralizada (p. ej. Audio Galaxy, eDonkey, Gnutella, Freenet).
- Arquitecturas orientadas a servicios (p.ej. servicios web).

Modelos de Sistema

- **Sistemas síncronos**:
  - El tiempo de ejecución de un paso de un proceso tiene límites superior e inferior conocidos.
  - El tiempo de transmisión a través de un canal de comunicación es acotado y conocido.
  - La deriva entre los relojes locales tiene una cota conocida.
  - Otra definición típica es en la que los nodos ejecutan pasos de computación en *lock-step*.

Modelos de Sistema

- **Sistemas asíncronos**:
  - Los pasos computacionales de los procesos se ejecutan tarde o temprano.
  - Los mensajes enviados por un canal son recibidos tarde o temprano.
  - Los relojes locales tienen derivas no acotadas.

Modelos de Sistema

- **Sistemas parcialmente síncronos**.
  - El tiempo de transmisión de los mensajes y ejecución de los pasos de los procesos tiene una cota pero es desconocida.
  - El sistema tiene dos periodos de funcionamiento. En el primero el sistema es totalmente asíncrono. En el segundo el sistema tiene una cota temporal conocida para la ejecución de procesos y transmisión de mensajes. El instante en el que el sistema pasa del periodo inicial al periodo permanente está acotado, pero es desconocido.

Modelos de Sistema

- Detectores de Fallos:
  - El sistema se amplía con un detector de fallos no fiable (puede cometer errores).
  - El grado de asincronía del sistema se modela con las garantías del detector de fallos.
  - Propiedades:
    - Precisión *(accuracy)*: No se sospecha de procesos correctos.
      - La precisión puede ser permanente o "tarde o temprano" (*eventually).*
      - Fuerte de ningún proceso correcto.
      - Débil: de algún proceso correcto.
    - Completitud *(completeness)*: Se sospecha de los procesos incorrectos.
      - Fuerte: Por todos los procesos correctos.
      - Débil: Por algún proceso correcto.

Modelos de Sistema

Accuracy / *Completeness*	***Strong***	***Weak***	***Eventually Strong***	***Eventually Weak***
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\Diamond\mathcal{P}$	*Eventually Strong* $\Diamond\mathcal{S}$
Weak	$\mathcal{Q}$	*Weak* $\mathcal{W}$	$\Diamond\mathcal{Q}$	*Eventually Weak* $\Diamond\mathcal{W}$

- El detector de fallos más débil que permite resolver consenso es $\Diamond$ W.
- $\Diamond$ W puede ser extendido con un pequeño algoritmo distribuido para obtener $\Diamond$ S

Modelos de Fallos

- Terminología:
  - Fallo *(failure)*: Desviación del comportamiento del sistema con respecto a su especificación.
  - Error *(error)*: La parte del estado del sistema que da lugar al fallo.
  - Defecto *(fault):* La causa del error.
- Los defectos (*faults*) pueden ser:
  - Transitorios: Duración limitada (p. ej. un fallo de la red).
  - Intermitentes: Defectos transitorios repetitivos.
  - Permanentes: Una vez el sistema falla debido a estos defectos ya no vuelve a funcionar si no se repara.

Modelos de Fallos

- Los defectos también se clasifican en función de su naturaleza:
  - De diseño: se originan durante la concepción del sistema o su actualización (p. ej. fallo del coprocesador matemático del Pentium).
  - Operacionales: Causados por causas físicas (p. ej. un fallo del ventilador de la CPU, causa un recalentamiento que a su vez hace que la CPU no se comporte de acuerdo a su especificación).
- Un sistema **tolera fallos** si a pesar del fallo de alguno de sus componentes el sistema sigue cumpliendo su especificación. Esto es, si es capaz de enmascarar fallos de sus componentes.

Modelos de Fallos

- Jerarquía de defectos (*faults*) que pueden ocurrir:
  - *Crash* (nodos *fail-silent)*. El sistema se cae pero mientras funciona respeta su especificación. Una vez caído no genera ninguna salida visible.
  - Omisión *(omission)*. Un componente omite alguna acción de forma eventual.
  - Temporización (*timing*). Un componente ejecuta una acción antes o después de lo especificado.
  - Arbitrarios o bizantinos (*Byzantine*). Cualquier tipo de fallo, incluidos los maliciosos (p. ej. los ocasionados por el recalentamiento de la CPU o por un hacker).

Modelos de Fallos

- Tipos de canales:
  - Fiables (no corrompen mensajes, no pierden mensajes, no duplican mensajes).
  - No fiables (pueden perder mensajes).
- Los canales fiables son una abstracción que se construye sobre canales no fiables justos (fair lossy).

Modelos de Interacción

- Paso de mensajes.
- Radiado *(multicast)*.
- Llamada a procedimiento remoto (RPC).
- Cita *(rendezvous)*.
- Cita multiflujo *(multithreaded rendezvous)*.
- Publicación/subscripción *(publish/subscribe)*

Paso de Mensajes

- No Fiable:
  - El mensaje puede llegar una vez, más de una vez, ninguna vez (p. ej. UDP).
- Fiable:
  - El mensaje llega exactamente una vez (p. ej. TCP).
- Ordenación FIFO:
  - Se garantiza que los mensajes se entregan al receptor en el orden en el que fueron enviados por el emisor (p. ej. TCP).

Paso de Mensajes

- Síncrono:
  - Envío síncrono: El envío no devuelve control hasta que el mensaje no ha sido recibido por el receptor.
  - Recepción síncrona: La recepción no devuelve control hasta que el mensaje no ha sido recibido.
- Asíncrono:
  - Envío asíncrono: Devuelve control sin esperar a que el mensaje haya sido enviado y recibido por el receptor.
  - Recepción asíncrona: Devuelve control sin esperar a que el mensaje haya sido recibido. Sólo cuando intenta accederse el contenido del mensaje se produce una sincronización (en el caso de que el mensaje no haya aún sido recibido).

Paso de Mensajes

- Mensajes no tipados:
  - Si el formato empleado por el emisor no coincide con el del receptor se produce un error de difícil localización (corrupción de los datos).
  - Ejemplo: sockets.
- Mensajes tipados:
  - Si los formatos de emisor y receptor no coinciden se eleva una excepción.

Paso de Mensajes

- Recepción selectiva (p.ej. Select de sockets):
  - Permite recibir de un canal cualquiera de entre un conjunto especificado en la instrucción de recepción selectiva, o de cualquier canal.
  - Un servidor admite mensajes enviados desde cualquier canal (de hecho, no conoce a los emisores a priori) y establece conexiones para cada mensaje de petición recibido lanzando un nuevo proceso o tarea (thread). La conexión queda establecida entre el cliente y el proceso/tarea que atenderá sus peticiones.
  - En ocasiones es posible establecer un timeout en para terminar la recepción si no se reciben mensajes en un intervalo de tiempo.

Radiado (Multicast)

- Interacción uno a muchos.
- Puede ser no fiable (p.ej IP-multicast) o fiable.
- Puede ofrecer distintas garantías de ordenación de mensajes (p.ej. Orden total).

Llamada a Procedimiento Remoto (RPC)

- Propuesto por Birrell & Nelson en 1984.
- Permite llamar a un servicio remoto de igual forma que a uno local.
- Tipado garantizado por el compilador como ocurre en las llamadas a procedimientos locales.
- Emparejamiento de llamadas garantizado.
- El sistema se encarga del aplanado/desaplanano de los parámetros (marshalling/unmarshallling):
  - A través de la generación automática de pares stub/skeleton.
- Inconveniente: El cliente queda bloqueado durante la ejecución del servicio.
- Proporciona transparencia de distribución (se llama de igual forma a un servicio local que a uno remoto).

Llamada a Procedimiento Remoto (RPC)

- La implementación puede adoptar distintas políticas de procesamiento de peticiones *(multithreading):*
  - Un proceso/thread para todas las peticiones de todos los clientes.
  - Un proceso/thread para todas las peticiones de un cliente.
  - Un proceso/thread para cada petición de cada cliente.
- Semánticas:
  - Al menos una vez (interesante para servicios idempotentes).
  - Como mucho una vez.
  - Exactamente una vez.
- Pueden ser para sistemas homogéneos o heterogéneos.

Cita o *Rendezvous*

- La cita es una evolución de la RPC (mecanismo de interacción propuesto en Ada 83 para la interacción entre tareas).
- Permite que el servidor realice procesamiento mientras el cliente progresa.
- Permite al servidor forzar un protocolo sobre un cliente único.
- Puede emplearse una instrucción de recepción selectiva.
- Puede asociarse un timeout a la recepción.
- Cuando una petición no puede satisfacerse se puede reencolar para terminar de procesarse posteriormente.

Multithreaded Rendezvous

- Permite imponer un protocolo de llamadas a distintos clientes de forma sencilla.
- Permite al servidor a realizar procesamiento sin bloquear al cliente.
- Permite recepción selectiva como el rendezvous.