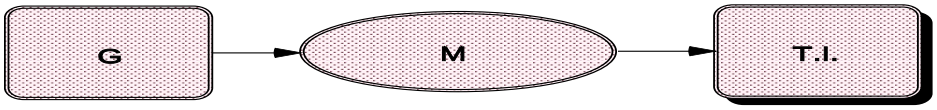
Complejidad y sistemas distribuidos.

El fenómeno del microprocesador está dando la vuelta al concepto de informatización que ha estado en boga durante muchos años. Estas pequeñas "pastillas" han puesto una gran potencia de procesamiento en manos de los usuarios, posibilitando capacidades hasta hace poco sólo reservadas a los miniordenadores o a los "mainframes".

Ahora, el reto consiste en coordinar recursos distribuidos, consiguiendo que colaboren eficientemente entre sí, para materializar de forma práctica y abordable su teórico potencial de servicios a los usuarios.

Interconectar ordenadores y lograr su cooperación no es tarea sencilla. Además de los problemas que hay que considerar respecto al hardware y al software, aparece el nuevo y complejo factor adicional de las comunicaciones. A fin de cuentas, la coordinación de recursos es un asunto simultáneo de control y comunicaciones. Por todo ello, los sistemas distribuidos se presentan como un caso claro de emergencia de complejidad.



1. Introducción.

Con la proliferación de redes de área local interconectando ordenadores, y su introducción en ámbitos tan variados como la oficina, la factoría de producción o los centros de investigación, se ha hecho posible pensar en la construcción de sistemas informáticos distribuidos. Comienza a hablarse, incluso, de sistemas distribuidos de área amplia, donde los recursos informáticos no solo estarían unidos por redes de área local, sino también por redes públicas de transmisión de datos[1]. Nuestro objetivo en este capítulo no va a ser tan amplio. Nos centraremos en lo que se suele denominar "sistemas distribuidos débilmente acoplados", en el ámbito de redes de área local[2].

2. Los sistemas distribuidos débilmente acoplados.

Estos sistemas van más allá de la mera compartición de ciertos elementos, como impresoras o servidores de ficheros. Pretenden conseguir que el usuario que interaccione con ellos perciba todos los recursos del sistema como locales, aunque en realidad los obtenga gracias a la colaboración de muchos elementos distribuidos. Esto permitiría una mayor dispersión geográfica de los elementos que colaboran para constituir el sistema. En pocas palabras, la idea es sustituir un gran "mainframe", del que cuelgan recursos centralizados, por una tupida red de pequeños y medianos ordenadores, conectados con una pléyade de recursos distribuidos.

Este cambio, si se realizara así, sin más, plantearía inmensos problemas al usuario. Cada vez que necesitase un servicio, debería saber dónde está, qué características tiene[3], cómo se accede a él, prever posibles conflictos con otros usuarios que también lo solicitasen... Y además, todo esto debe

---

1.- Un ejemplo de estos sistemas, aunque aún poco desarrollado y quizás algo sencillo, es el que nos ofrecen los bancos mediante los cajeros automáticos. Cuando nos acercamos a sacar dinero o a consultar nuestro saldo, el terminal nos proporciona la ilusión de que somos el único usuario de un sistema local. Sin embargo, la realidad es muy otra: para que podamos ver los últimos movimientos impresos en un papel, han debido colaborar eficientemente ordenadores situados quizás a varios cientos de kilómetros de distancia.

2.- En [Coulouris, 1988] se definen los sistemas distribuidos débilmente acoplados como "aquéllos donde los recursos compartidos necesarios para conseguir un servicio informático integrado son aportados por algunos de los ordenadores de la red, y son accedidos por software que corre en todos los ordenadores, usando la red para coordinar su trabajo y para transferir datos entre ellos".

3.- Usualmente se pretende que en los sistemas distribuidos los recursos informáticos no tengan por qué ser de características homogéneas. Únicamente han de soportar protocolos normalizados de acceso a la red, y poco más (al menos, en cuanto a requisitos hardware). La idea es que puedan interconectarse sin problemas incluso equipos de diferentes fabricantes, con diferentes arquitecturas.

ser hecho de una forma eficiente para no malgastar recursos (y aportar un rendimiento al menos similar al de las máquinas centralizadas, a un coste igual o menor). Aún más complicado será el caso en que el servicio no sea dado por un único recurso, sino que necesite de la colaboración de varios (que es el caso más habitual)[4]. Si no dotamos a estos sistemas de algún mecanismo capaz de absorber toda esta complejidad, probablemente nuestro sufrido usuario preferirá, con toda razón, un sistema centralizado. Y los mismos problemas encontrará un programador que intente diseñar software distribuido.

Como breve adelanto, digamos que estos mecanismos existen (o están en fase de investigación y desarrollo). Los usuarios dispondrán de sistemas operativos distribuidos que oculten todos los aspectos "internos" del sistema, ofreciendo una interfaz homogénea (de una forma parecida a como un sistema operativo tradicional lo hace para una máquina aislada). Los programadores pueden ayudarse de herramientas CASE para el diseño distribuido, y lo que es más importante, de modelos teóricos que podrán aplicar para simplificar sus diseños.

3. Categorías de complejidad en sistemas distribuidos.

Pero antes de entrar con algo más de detalle en estas soluciones, analicemos el problema básico de la complejidad de los sistemas informáticos, desde el punto de vista de la comunicación entre máquinas espacialmente separadas. Podemos distinguir tres categorías o niveles jerárquicos de complejidad relacionados con su diseño y operación:

a. **Complejidad interna**. Caen aquí los problemas de diseño de software para máquinas aisladas. Se trata con planificadores de procesos, gestores de memoria, etc. Son las tareas típicas de los **sistemas operativos para máquinas aisladas** (esto es, que no se comunican con otros ordenadores)[5].

b. **Complejidad de comunicación**. Cuando los ordenadores comienzan a "hablar entre sí", aparece un nuevo nivel de complejidad. El entorno donde aparece este nivel suele estar compuesto por máquinas individuales conectadas mediante enlaces de baja velocidad. Aparecen dos conceptos muy importantes: la asincronía (ya no tenemos un reloj común para todos los procesos) y la posibilidad de fallos locales (puede caerse una máquina, con

[4].- Un acto en principio sencillo, como puede ser el caso de un usuario que manda un mensaje a otro a través del correo electrónico, necesita de la colaboración de, al menos, tres procesos (que, en un entorno distribuido, normalmente no estarán en la misma máquina). El primero será el que interaccione con el creador del mensaje (editor de textos, o algo parecido), y se encargue de enviarlo al proceso repartidor. Éste, según la dirección, lo enviará a la máquina correspondiente. Por último, cuando el destinatario quiera leerlo, le hará falta un nuevo proceso que sea capaz de acceder al mensaje, y se lo presente de una forma adecuada.

[5].- En [Coulouris, 1988] se citan sistemas operativos centralizados como CTSS, Multics o Unix, que en su configuración más sencilla (sin la parte de comunicaciones), entran perfectamente dentro de esta categoría.

todos los procesos que en ella corren, mientras el resto del sistema sigue funcionando[6]. Históricamente, en este nivel hemos tenido (y tenemos) extensiones de los sistemas operativos clásicos, que incluyen mecanismos de comunicación[7].

c. **Complejidad de colaboración**. Ahora, los ordenadores no sólo hablan entre sí, sino que han de colaborar de forma que el usuario (ya sea una persona o un proceso informático) perciba el sistema como una entidad única y local (aunque en realidad todos los recursos estén distribuidos). El escenario sobre el que este concepto está comenzando a tomar forma está compuesto por máquinas relativamente pequeñas (típicamente estaciones de trabajo y servidores[8]), conectadas mediante una red local de alta velocidad (al menos, 10 Mbit/s). La necesidad de unas buenas primitivas de comunicación, que hagan uso eficiente de la red, se hace imprescindible. Se acentúan las características generales del nivel anterior. Y además emergen otras nuevas, debidas a la necesidad de colaboración. Problemas como la compartición de datos por procesos remotos o la ejecución en paralelo en máquinas distintas no hacen más que aumentar la complejidad. Para actuar en este estrato aparecen los **sistemas operativos distribuidos**[9]. Si bien suelen guardar diferentes grados de compatibilidad y similitud con los de máquina aislada, constituyen esencialmente una nueva forma de entender los sistemas operativos. Será a este nivel al que nos dedicaremos con más amplitud en lo que queda de apartado.

Cada uno de estos niveles implica un salto cualitativo en cuanto al tipo de complejidad que implica. Además, el que en un sistema aparezcan características de complejidad de un nivel dado no excluye

---

6.- Aparecen también otros problemas, como el control de acceso (necesidad de identificar la máquina con la que hablamos, para permitirle acceso sólo a los recursos a los que tiene derecho), la saturación de enlaces de comunicación y otros problemas de comunicaciones,...

7.- Por ejemplo Unix, que nació como sistema operativo "de máquina aislada" (prácticamente carecía de medios de comunicación con otras máquinas), fue extendido en Berkeley con mecanismos de comunicaciones, que permiten abrir canales de transferencia de información con máquinas remotas (*sockets*, protocolos TCP/IP, etc.).

8.- Por ejemplo, en [Mullender, 1990] se describe la arquitectura sobre la que se construye Amoeba (un ejemplo de sistema operativo distribuido). Supone la existencia de estaciones de trabajo (donde trabajarán los usuarios humanos), armarios de procesadores (que proporcionan la mayor parte de la potencia de cálculo), servidores especializados (de ficheros, de bases de datos, de impresión,...) y máquinas de comunicaciones ("gateways", para enlazar con otros sistemas remotos), unidos por una red de área local (normalmente, de tipo Ethernet).

9.- En los últimos años (especialmente a partir de 1980), muchos equipos han desarrollado sistemas operativos distribuidos. Como rápido exponente de estos esfuerzos, se incluye a continuación una lista (incompleta, por supuesto), tomada de [Coulouris, 1988]: Xerox DS (Xerox PARC, 1977), CDCS (Universidad de Cambridge, 1979), Locus (UCLA, 1980), Apollo Domain (Apollo Co., 1980), Newcastle Connection (Universidad de Newcastle, 1980), Grapevine (Xerox PARC, 1981), V-system (Universidad de Stanford, 1982), Argus (MIT, 1 983), Amoeba (Universidad Libre de Amsterdam, 1984), Unix+Sun NFS (Sun Micro., 1985), Mach (CMU, 1986), Chorus (Chorus systèmes, 1988),...

que también presente problemas característicos de los niveles inferiores. Por el contrario, puede entenderse que los niveles se estructuran de forma que cada uno engloba a los anteriores[10].

La **estructuración en tres niveles de las tecnologías de la información** que puede encontrarse en anexo sobre ofimática concuerda bastante bien con la ofrecida aquí para los sistemas informáticos[11]. El nivel de proceso de información se relaciona con lo que hemos llamado "complejidad interna", el de comunicación con la "complejidad de comunicación" y el de colaboración con la "complejidad de colaboración". Sin embargo, conviene no olvidar que en el enfoque actual no se ha tenido en cuenta (o casi) la incidencia del usuario humano sobre el sistema, ni la de los elementos que lo componen por sí solos. Por tanto, estamos dentro del ámbito de la **complejidad sistémica** del modelo de Sáez Vacas.

También podemos situarnos dentro del marco propuesto por Bell y Newell. Si recordamos, había un nivel donde las estructuras relevantes eran los ordenadores, las redes telemáticas, etc. Lo llamábamos (en realidad, lo llaman así Bell y Newell) **nivel PMS**. Ese es el ámbito en el que nos movemos cuando hacemos las distinciones que por el momento vamos exponiendo en este apartado.

4. Algunas soluciones.

Una vez descrito el escenario donde nos encontramos, pasemos a discutir brevemente algunas soluciones (a modo de ejemplo) propuestas para los problemas que plantea la complejidad de colaboración en el ámbito de los sistemas distribuidos. No perdamos de vista el hecho de que lo que buscamos (de nuevo desde el **esquema $H \times I \times O = IO$**) son interfaces *I* que permitan al observador *H*[12] manejar una imagen del objeto *IO* más sencilla que el objeto inicial *O* (en nuestro caso, un sistema distribuido).

4.1. El sistema Amoeba.

El **sistema operativo Amoeba** puede ser un buen ejemplo de este tipo de interfaz (entre los otros muchos sistemas operativos distribuidos en fase de desarrollo, o incluso ya comercialmente disponibles). Permite a un usuario (ser humano o proceso) manejar con relativa facilidad "objetos

---

10.- Por ejemplo, al diseñar un sistema en el ámbito de la complejidad de comunicación, seguiremos teniendo también problemas de complejidad interna.

11.- Y esto no es, por otra parte, extraño. Piense el lector que un sistema ofimático es, entre otras muchas cosas, un caso particular de sistema distribuido.

12.- Como ya hemos adelantado en algún momento, el "observador" no ha de ser forzosamente humano. Aquí se nos presenta un caso en que puede no serlo: cuando, por ejemplo, es un proceso informático, que pretende acceder a algún tipo de servicio distribuido.

distribuidos"[13], mediante operaciones del tipo "llamada a procedimiento remoto" (que será descrito brevemente más adelante). Actúa por lo tanto como **filtro de variedad** entre la del sistema distribuido (muy grande) y la del usuario, al limitar los tipos de interacciones posibles entre ambos a las permitidas por los objetos distribuidos disponibles. Para la especificación y uso de estos objetos Amoeba proporciona el AIL[14], que permite realizar estas tareas de forma sencilla. La seguridad viene dada por el uso de "capabilities", objetos abstractos asociados a las operaciones sobre objetos. El sistema da "capabilities" a los usuarios de forma que se asegure la protección del objeto ante usos negligentes o no autorizados. Alrededor de estos dos conceptos, objetos y capabilities (y con la ayuda de unos eficientes protocolos de comunicaciones), se construye el sistema Amoeba. El resultado es un sistema operativo que consigue que para el usuario sean transparentes aspectos como la localización física de los ficheros, la concurrencia de accesos sobre ellos, la replicación de datos o cierto grado de tolerancia frente a fallos[15].

4.2. Las llamadas a procedimiento remoto.

Al principio de este capítulo se habló de la importancia de disponer de herramientas conceptuales para facilitar el diseño de sistemas distribuidos. En lo que queda de él, vamos a describir dos de ellas, para que el lector pueda entender a qué se hacía referencia con esas palabras.

Ya hemos mencionado las **llamadas a procedimiento remoto** (RPC)[16] al hablar de los mecanismos básicos de Amoeba. La idea que sustenta este modelo es extender al ámbito de los sistemas distribuidos la forma "clásica" de solicitar un servicio: la llamada a procedimiento. Se pretende, por tanto, disponer de una interfaz que permita al programador llamar a un procedimiento, pasándole los parámetros de entrada y salida oportunos, sin preocuparse de la localización física donde se ejecutará. El sistema deberá, por lo tanto, descubrir si el procedimiento puede ejecutarse localmente. En caso contrario tendrá que localizar una máquina donde sí se pueda, y comunicarse con ella (pasando y recibiendo los parámetros adecuados). Y todo esto de forma transparente para el usuario. En un caso más general puede ser necesaria la colaboración de varios procesos, localizados posiblemente en máquinas distintas, para ejecutar el procedimiento demandado[17].

13.- En [Mullender, 1990] se describe un objeto Amoeba como "un conjunto de datos sobre los cuales los usuarios autorizados pueden realizar ciertas operaciones, independientemente de la situación física del usuario y del objeto".

14.- AIL significa "Amoeba Interface Language" (lenguaje de interfaz con Amoeba).

15.- Desde luego, estas breves líneas no pretenden ser una descripción exhaustiva del sistema Amoeba. El lector interesado puede encontrarla en [Mullender, 1990].

16.- En inglés, RPC (remote procedure call). Para más información sobre las RPC puede consultarse [Birrell, 1984].

17.- Éste es el caso, por ejemplo, de muchos de los servicios del sistema Isis, como el gestor de transacciones o el servicio de noticias (de una forma breve, puede decirse que el sistema Isis es "una extensión al sistema operativo Unix que proporciona una caja de herramientas para facilitar el diseño y la codificación de sistemas distribuidos"). Una información más exhaustiva sobre las características de Isis puede encontrarse en [Birman, 1990].

Desde el punto de vista de la complejidad, las llamadas a procedimiento remoto proporcionan un instrumento que reduce la complejidad que percibe el programador. Una vez construido todo el sistema de llamadas remotas (cosa que, si se hace bien, tendrá una utilidad general), el diseño de un sistema distribuido puede ser tratado como si fuera centralizado. Cuando se necesiten recursos de otras máquinas, llamaremos a los RPCs que gestionen estos recursos, de la misma manera que llamaríamos a un procedimiento local, olvidándonos de todos los problemas de comunicaciones, colaboración entre procesos remotos, etc[18]. Por tanto, constituyen un *I* que permite que el observador perciba un *IO* mucho más sencillo que el *O* original (todo el proceso de comunicaciones y ejecuciones remotas que implementa una RPC). Como modelo conceptual, si el sistema nos "obliga" a realizar todas las interacciones con otras máquinas a través de RPCs, el conjunto virtualmente infinito de posibles formas de realizar las comunicaciones queda reducido a un único elemento: la llamada a procedimiento. Al realizarse todas las interacciones entre máquinas según este modelo, el programador tiene que tener en cuenta menos posibilidades y combinaciones.

4.3. El espacio de tuplas de Linda.

El **espacio de tuplas de Linda**[19] proporciona otro mecanismo capaz de facilitar cierto tipo de distribución al programador. Consiste éste en una memoria virtual (organizada como espacio de tuplas), sobre la que se construyen tres operaciones básicas[20]:

a. *out*: añade una tupla al espacio.
b. *in*: lee una tupla del espacio, y la borra.
c. *read*: lee una tupla sin borrarla.

En realidad, todo el espacio de tuplas está replicado en las memorias locales de los procesadores. Pero el sistema se encarga de que las puestas al día sean hechas de forma automática (con todas las comunicaciones y transferencias de información necesarias para ello) y transparente para el programador. Se dispone por lo tanto de una memoria que aunque está distribuida y es vista de forma consistente por todos los procesadores, puede ser tratada de una forma sencilla, similar a como se hace con una memoria local.

De nuevo, el programador que usa Linda ve reducida la complejidad, al disponer de un modelo *IO* (el espacio de tuplas) que le permite manejar el objeto *O* (una memoria distribuida y replicada) de

---

18.- En realidad, no pueden olvidarse totalmente esos problemas. Por un lado, al ir las llamadas por la red, percibiremos un retardo mayor que en el caso de llamadas locales (que se realizan en memoria). Y por otro, el hecho de utilizar recursos remotos da lugar a algunas dificultades si, por ejemplo, alguna máquina se cae mientras ejecutaba una tarea para nosotros. Además, en algún momento ha habido que diseñar los RPCs...

19.- Linda S/Net es un sistema operativo desarrollado por AT&T, sobre un hardware especializado en comunicaciones tipo "broadcast" (comunicación de uno a muchos) rápidas. Una descripción detallada del sistema puede encontrarse en [Carriero, 1986].

20.- La descripción de las operaciones sobre las tuplas está tomada de [Birman, 1988].

una forma muy sencilla. El sistema funciona, por tanto, como una interfaz *I* (según el modelo de Sáez Vacas).

5. Resumen.

En este capítulo se ha pretendido aportar una rápida visión de varios aspectos del extenso mundo de los sistemas distribuidos, centrándonos en el ámbito de los débilmente acoplados en red local, por ser los más extendidos hasta el momento. La intención que nos ha movido a ello ha sido la de analizar sus **problemas de complejidad** (y alguna de las aproximaciones empleadas para tratar con ellos). Desde luego, el catálogo no pretende ser exhaustivo, y los problemas tratados deben ser considerados sólo a modo de ejemplo.

Cuando estudiamos estos sistemas, nos enfrentamos a casos de **complejidad sistémica**. Para tener un punto de referencia, se ofrece un sencillo marco donde situar y clasificar los problemas que aparecen, basado en la distinción de **tres categorías diferentes de complejidad**.

En una última parte, se hace un somero repaso de algunos ejemplos de soluciones ya empleadas (o en fase de investigación) en este campo. Concretamente, los casos tratados son el **sistema operativo Amoeba**, las **llamadas a procedimiento remoto** y el **espacio de tuplas de Linda**.

Bibliografía.

Dividida en dos partes. En primer lugar, Notas Bibliográficas, donde se describen los trabajos consultados más relevantes sobre el tema. Después, Referencias Bibliográficas, donde pueden encontrarse todas las citas utilizadas en el capítulo.

Notas bibliográficas.

El libro [Coulouris, 1988], ofrece una amplia perspectiva del campo de los sistemas distribuidos, si bien está orientado específicamente hacia los sistemas operativos. En él pueden encontrarse exposiciones bastante detalladas de la mayoría de los conceptos tratados en el capítulo.

Por otra parte, el ejemplar de mayo de 1990 del IEEE Computer, dedicado por entero a sistemas operativos, incluye muchos artículos de interés para quién desee conocer los esfuerzos de investigación que se están desarrollando hoy en día sobre este particular. Se tratan con especial interés los sistemas operativos distribuidos.

Referencias bibliográficas.

Birman, K.P. y Joseph,T.A (1988): "Exploiting replication", **Lecture notes from "Artic 88, an advanced course on operating systems, Trömso, Norway, July 5-14"**.

Birman, K.P. et al. (1990): **The ISIS system manual, version 2.0**, the ISIS Project, Cornell University.

Birrell, A. y Nelson, B. (1984): "Implementing remote procedure call", **ACM Transactions on Computer Systems**, vol.2, núm.1, febrero, pág.39-59.

Carriero, N. y Gelertner, D. (1986): "The Linda S/Net's Linda Kernel", **ACM TOCS**, vol.4, núm.2, mayo, pág.110-129.

Coulouris, G.F. y Dollimore,J. (1988): **Distributed systems**, Addison-Wesley.

Mullender, S.J. y van Rossum, G. (1990): "Amoeba: a distributed operating system for the 1990s", **IEEE Computer**, vol.23, núm.5, mayo, pág.44-53.