UNIVERSIDAD DE ALCALÁ DE HENARES

CURSO 1998/1999

DISEÑO DE APLICACIONES DISTRIBUIDAS EN CONTROL

1 Introducción

1 Introducción a los sistemas distribuidos

2 Características principales

3 Ventajas e inconvenientes

2 objetivos de diseño

2.3 el modelo cliente−servidor

1 clientes y servidores

2 Un ejemplo cliente−servidor

3 Direccionamiento

4 Primitivas con bloqueo vs sin bloqueo

5 Primitivas almacenadas en buffer vs no almacenadas

6 Primitivas confiables vs no confiables

7 Implantación del modelo cliente−servidor

2.4 Llamada a un procedimiento remoto (RPC)

1 Operación básica de RPC

2 Transferencia de parámetros

3 Conexión dinámica

4 Semántica de RPC en presencia de fallos

5 Aspectos de la implantación

2.5 Comunicación en grupo 99

1 introducción

2 Aspectos de diseño

2 Estructura del sistema

3 Ocultamiento

4 Réplica

5 Un ejemplo: el sistema de archivos de red (NFS) de Sun

6 Lecciones aprendidas

5.3 Tendencias en los sistemas distribuidos de archivos

TEMA 1. INTRODUCCIÓN

1.− Introducción a los sistemas distribuidos

Hoy en día los sistemas de computo están organizados por varios ordenadores conectados en red, esto es un sistema distribuido. El problema que se plantea es que es necesario un software para coordinar las actividades. Una colección de sistemas independientes conectados con una red con un software diseñado para proporcionar soluciones de computo integradas ! Sistema distribuido.

Requerimientos:

- Seguridad Suelen tener BD centralizados
- Disponibilidad ! y redes dedicadas para mejorar
- Privacidad el tiempo de respuesta

UNIX se diseñó como un sistema multiusuario, y posteriormente se hizo el UNIX distribuido permitiendo la comunicación entre procesos.

2.− Características principales

- ***Compartición de recursos***

Tanto en SW como en HD los beneficios de estos servicios son en cuanto software, compartir datos, y hardware ahorro de dinero. Se denomina gestor de recursos al módulo software que gestiona recursos del mismo tipo. El resto de recursos se comunica con el gestor de recursos para utilizar otros recursos. Existen dos modelos de diseños distribuidos.

- Modelo Cliente/Servidor. El servidor gestiona los recursos que demanda el cliente. Un mismo proceso puede ser cliente y servidor. Puede ser cliente de un recurso y servidor de otros. El cliente solicita el recurso al servidor y si es válido el servidor se lo concede y le responde de su validez.

Servicio es el recurso que se da a los clientes, es una entidad abstracta y es proporcionado por un conjunto de servicios que corren en distintas máquinas conectadas, servicio distribuido.

Servidor es el que presta el servicio.

- Modelo basado en objetos: Cada recurso es visto como un objeto y este objeto tiene un identificador unívoco que le permite moverse en la red sin variar su identidad. Los beneficios son la sencillez la flexibilidad. Los recursos son vistos de forma uniforme.

- ***Apertura***

Determina en que medida el sistema es ampliable y la capacidad de añadir recursos compartidos sin interrumpir servicios. Esto se hace con la normalización de interfaces. Requerimos que el sistema sea extensible tanto de manera software como hardware. A nivel HD permitiendo que se añadan nuevos ordenadores al sistema y a nivel SW permitiendo añadir nuevos servicios

- ***Concurrencia***

Cuando existen nuevos procesos en el ordenador se dice que se están ejecutando concurrentemente. Si sólo tenemos un procesador se produce una multiplexación temporal, si tenemos n procesadores tengo paralelismo y podemos ejecutarlos simultáneamente. En un sistema distribuido tendremos paralelismo. La concurrencia surge porque los usuarios pueden estar utilizando distintas tareas. Los accesos concurrentes de recursos deben ser sincronizados.

- ***Escalabilidad***

Operan eficientemente en diferentes escalas. Podemos tener desde 2 ordenadores hasta cientos, pero en estas diferentes escalas deben funcionar eficientemente. Que sea escalable un sistema es complicado. Un sistema distribuido debe ser diseñado de forma que ningún recurso ni de SW ni de HW sea restringido.

La escalabilidad en sistemas distribuidos supone que a veces hay que hacer varios recursos para ello. Si estamos compartiendo un fichero y lo vamos modificando debe reflejarse a los diferentes usuarios.

- ***Tolerancia a fallos***

Se dice que un sistema es fiable si cumple:

  - Seguridad: ante accesos no deseables
  - Consistencia: a la hora de acceder a los mismos datos

Tolerancia a fallos: ocultar los fallos al usuario. Los fallos pueden ser HD y SW. Los fallos HD se solucionan con duplicación HD, lo que supone un coste económico alto, con lo que nos lo planteamos sólo en los sistemas críticos, es decir, en los servidores ! Servidor de respaldo.

En cuanto a los fallos SW se debe recuperar la situación inicial antes de hacer comenzado el proceso que produjo el fallo. La disponibilidad en un sistema es la proporción de tiempo que está libre para su uso. Si tenemos un sistema multiusuario, el fallo de un usuario puede hacer que caiga el sistema, en cambio en un sistema distribuido sólo hará que falle donde se produjo el error ese usuario. Si la red se cae hace que caiga todo el sistema. Luego el punto crítico está en la red.

- ***Transparencia***

Es la ocultación que se proporciona al usuario y a los programadores de aplicaciones de los recursos del sistema. Es el grado de concurrencia del usuario sobre la composición del sistema. El usuario lo concibe como un todo, no como un conjunto de componentes independientes. La separación de componentes proporciona ventajas como:

  - Ejecución en paralelo
  - Tolerancia a fallos

ISO identifica 8 niveles de transparencia de red:

- Transparencia de acceso: el acceso es igual de manera remota y local. No es consciente de donde está localizado el recurso.
- Transparencia de localización: el acceso a los recursos es independiente de la localización física del recurso.
- Transparencia de concurrencia: permite la ejecución concurrente sin interacción entre procesos.
- Transparencia de fallos: Oculta fallos parciales tanto en el SW como en el HW.
- Transparencia de replicación: permite replicaciones de componentes para incrementar la fiabilidad y rendimiento del sistema.
- Transparencia de migración: permite cambios en la ubicación de los recursos.
- Transparencia de rendimiento: permite reconfiguraciones del sistema cuando varía la carga de trabajo.
- Transparencia de escala: las aplicaciones se reconfiguran ante un cambio de tamaño.

Un email si tiene transparencia de red.

Hay ocasiones en la que es interesante que no hay transparencia, por ejemplo a la hora de imprimir.

3.− Ventajas e inconvenientes de un sistema distribuido

- ***Ventajas de un SD con un S. Centralizado***
- Económica. Nos salen más baratos 100 PC's y una buena máquina que una máquina de la hostia. Proporciona una mejor relación precio/rendimiento, y un mayor rendimiento que un sistema centralizado.
- Existen aplicaciones que por su naturaleza son distribuidas (pe. juego DOOM).
- Confiabilidad o tolerancia a fallos.
- Crecimiento de forma lineal permite incrementos progresivos con pequeñas inversiones.

Se puede decir que a largo plazo la primera fuerza motriz de los sistemas distribuidos es la existencia de un gran número de ordenadores personales y la necesidad de que las personas trabajen juntas y compartan información.

- ***Ventajas de un SD con un PC independiente***
- La compartición de datos (reserva de un billete).
- Compartir recursos caros (impresoras láser).
- Comunicación entre las personas (email).
- Mayor flexibilidad, el trabajo se distribuye en todas las máquinas aprovechando la máxima efectividad.
- ***Inconvenientes***
- El SW ! elegir el SO. Los lenguajes de programación y aplicación más convenientes para este sistema. El conocimiento que deben tener los usuarios sobre la distribución del sistema. ¿Qué debe hacer el sistema? Y ¿Qué debe hacer el usuario?
- Redes de comunicación. Las redes pueden perder mensajes, luego necesitamos un SW que pueda detectar y recuperar estos mensajes.
- Seguridad en el acceso a los datos, no todo el mundo tiene que tener acceso a cualquier dato. Hay que tener un control sobre el acceso a datos, si estos son públicos o privados.

TEMA 2. OBJETIVOS DE DISEÑO

Los objetivos de diseño pueden ser expresados mediante lo que se llama garantías, estas son afirmaciones validables y verificables sobre el comportamiento del sistema. El grado de comportamiento de dichos objetivos nos dará una idea sobre la adecuación del diseño. Los principales problemas técnicos son:

- Direccionamiento. Los SD se basan en la compartición de recursos y en su transparencia, dado lo cual los nombres que se asignan a los recursos deben tener un significado global independiente de su localización. Estos nombres deben ser soportados por un sistema de compartición de nombres que realiza la

interpretación de esto. Habrá que diseñar un esquema de direccionamiento escalable y eficiente.

- Comunicación. Las técnicas empleadas deben ser fiables y con buenas prestaciones. Un principio de diseño es la optimización de prestaciones.
- Estructura del SW. Es muy importante que se realice con interfaces bien definidos basados en estructuras de datos. Os servicios pueden ser vistos como gestores de objetos y la interfaz a un servicio puede ser vista como un conjunto de operaciones. Habrá que estructurar el sistema de manera que se puedan introducir nuevos servicios, esto se debe hacer sin tener que duplicar servicios que ya existan y hacerse también de forma integrada con los que ya existen.
- Distribución. Se persigue conseguir buenas prestaciones y una distribución equilibrada de la carga de trabajo entre el conjunto de procesadores disponibles para evitar la sobrecarga en unos recursos y la infrautilización de otros.
- Consistencia. Para mantenerla se pueden tener problemas de prestaciones, hay que mantenerla con unos costes aceptables.
- ***Direccionamiento***

Todo recurso debe tener un nombre o identificador unívoco. Para que un proceso acceda a un recurso externo debe conocer su nombre.

Nombre ! nombre utilizado por usuario o por programa

Identificador ! nombres que pueden ser interpretados o utilizados por programas

Un nombre es resuelto o traducido cuando es convertido a un formato que permite invocar o solicitar una acción del recurso a que se refiere. En los SD, traducir un nombre implica que se genera un identificador y una serie de atributos útiles para la comunicación. El tipo de identificador de comunicaciones depende del sistema de comunicaciones (pe Internet ! IP+puerto). Puede ser a varios niveles, en cada nivel el nombre es traducido al nombre del nivel inferior que permite la comunicación con otro recurso, que permite la comunicación SW del mismo nivel. Se genera un identificador de comunicaciones que es entendido por el nivel de comunicación y por un gestor de recursos.

La traducción de nombres en un SD toma las siguientes consideraciones:

- Se debe reservar un espacio de nombres adecuado para cada tipo de recurso, puede ser finito o potencialmente finito, puede ser estructurado o plano, sencillo o compacto pero todos los recursos deben tener nombres diferentes que se identifiquen independientemente de su localización.
- Todos los nombres deben ser traducidos a identificadores de comunicación mediante:

- Servidores de nombres, es un servidor que mantiene copia de nombres y de su traducción a identificadores de comunicaciones.
- Gestor de recursos, el que hace la traducción
- Mantener los nombres más recientemente utilizados en una caché local, de esta forma se evitan comunicaciones innecesarias con los servidores o con el gestor de recursos. Los nombres son siempre traducidos en relación a un contexto. En los sistemas de ordenadores, estos contextos se representan por BD o tablas de nombres (ej. directorio es el contexto de un fichero). Por tanto para traducir un nombre hay que traducir el nombre y su contexto.

Un servidor de nombres traduce los nombres a identificadores en otro espacio de identificadores. Además, también registra nuevos miembros e identificadores. También puede borrarlos, de esta forma, gestionamos una BD de sus nombres e identificadores para cada contexto, estos servidores de nombres son una parte esencial de los SD. Hay muchos tipos de nombres e identificadores:

- Sencillos: son diseñados para poder ser leídos por los humanos.

Compactos: tienen una representación compacta en memoria o de manera que dé una serie de pistas de la localización del objeto que representa (este último debe ser evitado debido a la transparencia)

- Jerárquicos: pueden tener una representación interna que representa su posición en un sistema jerárquico.

Planos: los nombres están formados por un conjunto plano de identificadores numéricos y/o símbolos. La ventaja de los jerárquicos es que cada parte del nombre puede ser traducido en su contexto y el mismo nombre puede ser utilizado en distintos contextos.

- Infinitos: los jerárquicos son potencialmente infinitos, permiten que el sistema crezca de manera ilimitada.

Finitos: los planos son generalmente finitos, su tamaño está limitado por la longitud máxima de los nombres, si no hay tamaño máximo, será infinito.

- Puros: son cadenas de bits sin ningún tipo de identificación.

El resto: el nombre lleva información sobre la localización (no recomendable) o los posibles usos del objeto. Pueden ser traducidos por algún algoritmo a través del nombre.

El esquema de nombres puede ser aprovechado para prohibir accesos no autorizados a los recursos. Para llevar un control de accesos no autorizados, el servidor de nombres debe llevar una comprobación de la identidad de los procesos que le solicitan la traducción de un nombre.

- ***Comunicación***

Un SD está formado por procesos distribuidos tanto en localización como lógicamente, por lo que deben comunicarse entre ellos para realizar tareas en común. La comunicación entre procesos:

- Transferencia de datos entre emisor−receptor compartiendo un canal de comunicación
- En algunas ocasiones una sincronización entre procesos de manera que el emisor o el receptor quede parado hace que el otro proceso haga una determinada acción.

Las tuberías de información son para enviar/recibir. A esto se le llama paso de mensajes entre procesos, y cada paso de mensajes origina la transmisión al emisor del mensaje, esta transmisión se hace por el canal o puerto, y la aceptación por el receptor del mensaje.

La comunicación del emisor puede ser:

- Síncrona o bloqueante, el emisor espera ha que el receptor lo reciba
- Asíncrona o no bloqueante, el emisor deja el emisor en una cola de mensajes y él sigue con lo suyo.

Esta comunicación también puede ser respecto al receptor. Normalmente la emisión es no bloqueante y la recepción sí.

Modelos de comunicación.

- Modelo cliente/servidor para la comunicación entre dos procesos
- Modelo multipunto entre un grupo de procesos que cooperan

Modelo cliente servidor. Está orientado hacia la provisión de un servicio, y normalmente un intercambio que

consiste en la transmisión de una petición del proceso cliente al proceso servidor, después se realiza el proceso y por último hay una contestación al cliente desde el servidor. Este tipo de conversación necesita el uso de dos mensajes y de una sincronización. El cliente normalmente se queda bloqueado desde que manda la petición hasta que recibe la respuesta. Un proceso servidor puede dar servicio a múltiples clientes y además no tiene porque conocerlos anteriormente. Cada petición lleva un identificador de la comunicación que es utilizado por el servidor a la hora de contestar. En un sistema abierto, no mantiene todos los identificadores posibles, sino que hace una designación dinámica, cuando un servidor arranca este se registra en un servidor de nombres, proporcionando su dirección de red y el nombre del servicio que proporciona. Los clientes lo que hacen es preguntar al servidor de nombres por el servicio requerido y este contesta con la dirección del servidor que dará ese servicio. Un cliente puede ser también servidor y viceversa.

Grupo multipunto. También utiliza la comunicación a través del paso de mensajes, pero el destino del mensaje no es un proceso, sino un grupo de procesos. A una operación de multipunto le corresponde una recepción en cada miembro del grupo de procesos, a esto se le llama comunicación en grupo o multipunto. No es lo mismo esto que la comunicación broadcast o difusión, en la que se envía el mismo mensaje a todos los ordenadores. Un ejemplo es para localización de un objeto, un cliente busca el nombre de un fichero mandando el mensaje a un servidor de ficheros, respondiéndole a este. También se puede utilizar para tolerancia a fallos, para comprobar el estado de un grupo de servidores También se utiliza multipunto para comunicaciones múltiples, por ejemplo mandar la actualización de la hora. Para realizar la comunicación multipunto puede existir o no soporte multipunto, el mensaje se envía una vez en paralelo a todos los miembros del grupo. Puede o no puede conocer el cliente o el servidor los procesos que forman parte de este grupo multipunto. Este grupo puede ser un grupo dinámico (hay altas y bajas).

- ***Estructura del SW***

En un sistema centralizado, los SO son monolíticos dado que la interfaz que dan a los programas de aplicación es única e incambiable. En un sistema distribuido, los SO no deben ser monolíticos dado que los servicios deben utilizar varios recursos y al revés, además, se debe poder extender los recursos ofrecidos a los programas de aplicación por medio de nuevos servicios.

Los niveles SW y HD de un sistema centralizado son:

- Aplicación Cada uno de estos niveles utiliza las facilidades
- Lenguajes de programación que proporciona el nivel anterior. El SO es el
- Sistema Operativo nivel SW más importante y es el que gestiona
- Hardware los recursos del sistema

SO ! Gestión de recursos básicos ! Administración de la memoria realizando tareas y gestión de procesos. Se encarga de la planificación de procesos y la gestión de periféricos. Proporciona servicios para aplicaciones, acceso y gestión de ficheros, control de reloj Se debe permitir que se puedan añadir nuevos servicios, con este fin, en un SD se restringen los recursos del núcleo a la gestión de recursos básicos y a la comunicación entre procesos. El núcleo de un SO está diseñado para no ser modificado normalmente, el núcleo de un SO se encargará de los procesos más básicos. Además en el SD se introduce un nuevo componente SW, los servicios abiertos, que no requieren acceso a los datos o al código del núcleo, por lo que no requieren que estén en el núcleo del SO.

Los servicios abiertos no requieren acceso a los datos y son aplicados al mismo nivel que los procesos de aplicación. Esta apertura permite que los sistemas sean configurados dependiendo de la necesidad de usuarios o aplicaciones. En un sistema abierto se pueden integrar diferentes ordenadores con distintas arquitecturas y/o SO.

Los usuarios de la aplicación tendrán una visión muy pequeña del sistema global, sin embargo, el

programador ve un SD y utilizará las funciones de los programas más que las cuentas de las máquinas, utilizará el soporte de programación distribuida y los servicios abiertos. Un administrador del sistema tendrá un conjunto de las máquinas del sistema. En la estructura del SD se pueden distinguir tres categorías de SW que aportan la ejecución de aplicaciones:

- Los servicios del núcleo del SO
- Los servicios abiertos
- Soporte de programación distribuida.
- En cada ordenador existe un núcleo del SO responsable de proporcionar los recursos básicos a la vez que proporciona la protección de los componentes HW básicos ante accesos no deseados (ej. 4BSD Unix). Muchos de los servicios ofrecidos por UNIX pueden ser también proporcionados por los servicios abiertos, lo que llevará a una duplicación de servicios, creándose micronúcleos que proporcionan el conjunto de recursos y servicios mínimo sobre el que se construye el resto de servicios. En un SD los diferentes ordenadores pueden tener diferentes micronúcleos siempre y cuando los servicios de comunicaciones estén estandarizados.
- Estos son los servicios que proporcionan la facilidad de programación de un SD. De esta forma, a los programas de aplicación se les pueden instalar nuevos servicios cuando se requiera, de manera que los SD se adapten a las necesidades de las aplicaciones. Van desde los recursos de ficheros hasta la facilidades que no suelen formar parte del SO (ej. email). La distinción entre servicios del núcleo y servicios abiertos es que el núcleo no puede ser abierto y debe estar protegido contra modificaciones en tiempo de ejecución, ya que se encarga de proporcionar una interfaz estable a los recursos de bajo nivel.
- Proporciona la facilidad de programación para que los programas escritor mediante lenguajes convencionales trabajen juntos (ej. RPC, permite pedir que se ejecute el proceso en una máquina remota y que devuelva el resultado del proceso).
- ***Distribución de carga de tiempo***

En un sistema centralizado, el procesador y toda la memoria son utilizados por SO dependiendo de la carga de trabajo que tengo; en un SD existe una serie de procesadores y espacio de memoria entre los que habrá que distribuir la carga de trabajo, esta partición debe realizarse de forma equilibrada de manera que evitemos una infrautilización. En este manejo de la carga hay problemas de heterogeneidad de los nodos y compartición de recursos entre usuarios. Existen varios modelos de trabajo:

- **Modelo de estación de trabajo**, aquí la arquitectura del SD está constituida por una serie de ordenadores interconectados en la que cada máquina pertenece a un usuario.

Cada usuario tendrá limitado el tamaño de tarea que podrá utilizar a la capacidad del procesador y de memoria de su ordenador.

Ventajas: sencillo, tiempo de respuesta garantizado, alto nivel de autonomía, no plantea problemas de consistencia.

Desventajas: no optimiza los procesadores ni recursos de memoria, máquinas inactivas y otras sobrecargadas, n permite que un usuario que necesite más memoria o capacidad de cómputo que la que proporciona su máquina pueda obtener más recursos. Existen dos modificaciones a este modelo para mejorarlo, como el modelo de pila de procesadores y el modelo de estaciones inactivas.

- **Modelo de pila de procesadores**. Es un sistema en el que se añade una pila de procesadores. Esta pila está formada por una serie de procesadores que se asignan dinámicamente baja demanda.

Estos procesadores suelen ser ordenadores baratos que tienen básicamente un procesador, memoria y una tarjeta de red. Cada ordenador es independiente. En este modelo, los procesadores son asignados a los procesos de manea permanente desde el inicio de ejecución hasta que termina, de manera que la compartición

de procesadores se hace a nivel de proceso.

La granularidad es a nivel de proceso, dado que el usuario puede utilizar varios procesadores simultáneamente, pudiendo tener mayor capacidad de computo, por ejemplo, la compilación de un programa multisegmento en la que cada segmento puede ser compilado independientemente, podemos compilar n segmentos en n computadoras, reduciendo el tiempo en n veces. En este modelo el usuario puede hacer varias si tiene una estación de trabajo pequeño o incluso un X−Terminal, que es un terminal gráfico sin terminal de red y sin disco, que ejecuta un sistema de e/s X−11 que maneja la pantalla y los dispositivos interactivos (teclado y ratón). Este servidor X−11 se ejecuta en la estación X−Terminal y se encarga de ejecutar su teclado, ratón y pantalla. Los clientes del servidor X−11 son los programas de aplicación con los que el usuario está interactuando, estos clientes gestionan los comandos y las entradas que proporciona el usuario y genera la información correspondiente que debe ser sacada por pantalla. Los programas cliente se comunican con el servidor X−11 proporcionando las librerías y procedimientos necesarios para crear ventanas y para visualizar y modificar objetos en las ventanas. No es necesario que estos clientes y servidores estén en la misma estación, la comunicación entre ellos se hace mediante RTC.

Los clientes X−11 suelen estar localizados en diferentes procesa− dores de la pila, así un X−Terminal puede tener el mismo rendimiento que tendría una estación de trabajo considerando que no necesite mucha comunicación que haga que pierda rendimiento (cuello de botella).

En el modelo de pila de ordenadores el terminal de usuario puede solamente proporcionar un acceso a los recursos del sistema, en este modelo las tareas de usuario se pueden ejecutar parcial o totalmente en la pila de procesadores. Cuando un usuario inicia una tarea se la asigna un ordenador de la pila donde se ejecuta la tarea, si el usuario inicia más de una tarea o esta se expande en subtareas, se pueden asignar varios ordenadores de la pila, uno para tarea, de esta forma las tareas se pueden ejecutar en paralelo. Un sistema que soporta la pila de ordenadores es Amoeba, en este sistema, el usuario puede utilizar programas de utilización iterativos o gráficos, permite ejecutar programas desde un X−Terminal o desde un procesador de menor capacidad que la requerida por el programa. También permite que algunos procesos servidores que no necesitan acceso directo a periféricos, como es el caso de un servidor de nombres, corran en la pila de ordenadores. En este sistema la asignación de los ordenadores de la pila es coordinada por un servidor de ejecución que realiza la asignación de forma dinámica. Cuando un programa cliente necesita ejecutar un programa, contacta con el servidor de ejecución, este le asigna un procesador y esta asignación la realiza dependiendo de la carga actual del resto de procesadores y de los requerimientos del proceso y memoria del programa. El programa se compila y linka para cada arquitectura, obteniendo distintos ejecutables para cada una de ellas (sistemas Plan y Clouds). En general las ventajas de pila de ordenadores son:

- Económicas
- Soporta diferentes arquitecturas
- Crecimiento por incremento
- Disminuyen los tiempos de respuesta

- **Modelo de estaciones inactivas**

En este modelo se usan estaciones poco cargadas o inactivas como una pila dinámica de ordenadores que pueden ser procesadas de manera análoga a la pila de ordenadores.

Muchas de las estaciones de trabajo están inactivas e infrautilizadas porque están fuera del horario de trabajo o con tareas que consumen pocos procesos como es el caso de la edición de documentos. Este modelo se basa en la capacidad de proceso de esas estaciones que puede ser utilizada para ejecutar taras de usuarios que están sobre estaciones que no tienen capacidad de utilizarlas. El modelo debe responder a tres preguntas :

- ¿Cómo encontrar una estación inactiva?

- ¿Cómo ejecutar en remoto de forma transparente?
- ¿Qué ocurre si regresa el usuario local?

Una estación inactiva es una estación en la que nadie está interactuando durante varios minutos. Si la distribución de carga es controlada por el servidor, cuando la máquina esté inactiva anunciará su inactividad, si la carga es controlada por el cliente, el cliente solicita ayuda y las máquinas inactivas le responden eligiendo una y configurándose dicha máquina. Un modelo válido será el Sprite.

Para la segunda pregunta se debe configurar el proceso remoto de la misma forma que el local, todo lo que tiene que ver con el árbol de archivos, directorio de trabajo y variables de entorno. Entre los problemas que tiene esta configuración de sistema remoto están las llamadas al sistema (ratón, teclado, pantalla).

Para responder la tercera pregunta se puede no hacer nada, con lo cual el usuario local pierde el tiempo de respuesta garantizado, os e puede eliminar al intruso, esta eliminación puede ser mediante una advertencia o brusca (se le echa sin más). En cualquier caso, la máquina se debe quedar como se encontraba inicialmente. El Sprite es un SO distribuido que proporciona una facilidad al usuario para que ejecute unos comandos con máquinas inactivas o infrautilizadas. Esta elección se produce de manera transparente al usuario de igual manera que en el sistema Moeba. Sprite proporciona facilidad para la migración de procesos que consiste en la recolocación de un programa en ejecución de una máquina a otra. Se puede migrar la ejecución de un programa a la que la lanzo cuando esta no puede seguir debido, por ejemplo, a la conexión de un usuario o porque se empieza a utilizar su capacidad de proceso de manera más activa.

- **Multiprocesadores de memoria compartida**

Permiten gran proceso mediante varios procesadores. Este tipo de ordenadores pueden ser utilizados en pila de procesadores o en modelo de estaciones inactivas. Existen diferentes arquitecturas en cuanto a memoria y diferentes esquemas.

Este tipo de ordenadores es muy utilizado en los servidores en SD dado que proporcionan gran capacidad de proceso tanto a nivel SW como HW. Un típico procesador de memoria compartida está formado por procesadores independientes que pueden ejecutar programas separados. Cada uno de estos procesadores independientes es equivalente en capacidad de proceso al procesador de una máquina convencional, de hecho, en la mayoría de los casos es el mismo. El número de procesadores de que consta un multiprocesador de memoria compartida varía de 2 a 64, este límite es debido a los costes de ingeniería y a la pérdida de rendimiento que se origina a la hora de conseguir interfaces eficientes a la memoria compartida, lo que origina tener una memoria caché grande para cada procesador. A través de la memoria compartida se realizan operaciones entre procesos a grandes velocidades ya que se pueden asignar los procesadores según los requisitos de las tareas de usuario. Este tipo de arquitecturas se suele utilizar para soportar a los servidores de procesos de forma que un ordenador multiprocesador pueda atender diferentes clientes en paralelo, reduciendo los cuellos de proceso que se suelen producir en los servicios claves en SD.

- ***Consistencia***

En un SD los problemas de consistencia vienen originados por la separación de los recursos de separamiento y por la propia concurrencia de los SD.

Existen varios tipos de consistencia: de actualización; replicación; caché; fallos; sincronización; interface de usuario.

Consistencia de actualización

Este problema se origina cuando los datos son adquiridos y actualizados de manera concurrente por varios

procesadores. La actualización de varios tipos de datos no se suele dar de forma instantánea pero debe aparecerlo al usuario, es lo que se denomina actualización de datos crónica, una serie de datos relacionados por un proceso aparecen al resto de procesos como si fueran instantáneos. Este problema es realmente importante en los SD dado que muchos usuarios acceden a datos compartidos y porque el funcionamiento del propio sistema depende de la consistencia de ciertas BD como son los directorios gestionados por los servidores de ficheros, o las BD de nombres.

Consistencia de replicación

Cuando un dato se ha cargado en varias máquinas y posteriormente se han hecho modificaciones, hay probabilidad de que hay problemas de consistencia entre las distintas copias. Por ejemplo un juego multiusuario, cada jugador mueve una criatura en un laberinto, cada jugador tienen un visión global del laberinto y de las criaturas. La replicación de cada movimiento se puede hacer mediante un envío multipunto desde el ordenador que realiza el movimiento al resto. Si dos jugadores se mueve simultáneamente, puede ocurrir que los jugadores vean los movimientos en distinto orden, pudiendo parecer que se han roto las reglas del juego. Otro ejemplo son las news de Internet.

Consistencia de Caché

Se denomina caching al mecanismo implementado por SW en el cliente que mantiene en el entorno del mismo una copia de los datos para su posterior reutilización, evitando que los envía de nuevo cuando se necesitan. Consiste en el almacenamiento local de un recurso para mejorar la eficiencia. Dado que el espacio de almacenamiento del cliente es limitado, sólo se almacena una parte de los datos, y este almacenamiento hace que los datos accedidos en el pasado cercano tienen mayor probabilidad de ser accedidos en el futuro inmediato. El caching puede ser implementado por una paquete SW en el cliente sin modificar el SD. El caching es transparente y mejora la eficiencia de e/s.

Ejemplo: un programa en C lee datos de un fichero, lee 1 byte/t en un intercambio de petición, y la respuesta lleva unos 2 ms en una red de 10 Mbits. En cambio, un mensaje de 1Kbyte/t de otro ordenador lleva 3 ms en las mismas peticiones, es decir, traer 1 Kbyte/t es una 700 veces menos costoso que traer 1 byte/unidad de tiempo.

Una vez que está en la unidad C, podemos acceder byte a byte de manera local. Si necesitamos acceder a los mismos datos mientras sigan almacenados, se podrán utilizar los problemas de consistencia cuando los datos son actualizados por otro usuario cuando los datos ya estaban almacenados localmente.

Ejemplo: cuando el mismo bloque de fichero es solicitado y almacenamos localmente este fichero, para los demás usuarios no coincide después de la modificación de alguno. Puedo mandar el bloque actualizado al servidor, y luego éste tendrá que mandarlo a los clientes para que conozcan las últimas modificaciones.

Consistencia de fallos

En un sistema centralizado, cuando un sistema falla, todos los programas fallan de manera simultánea. Se dice que sólo hay un modo de fallo en un SD, el fallo de un componente no implica el fallo total, si falla el componente el resto con los que cooperaba seguirán funcionando, pero si estos dependen del ordenador que ha fallado, entonces fallarán posteriormente porque dependen del ordenador que ha fallado. Hay múltiples modos de fallo en un SD. Cada uno de los componentes puede ser que falle posteriormente en un punto distinto del programa con el objeto de que todos los componentes mantengan la consistencia. Se necesitan procesos de eliminación de forma que todos los datos vuelvan al estado anterior al fallo, esto que puede suponer pérdida del trabajo realizado.

Consistencia de sincronización

Muchos de los algoritmos dependen de relojes internos porque su funcionamiento dependen del tiempo en el que ocurrió un evento. En un SD una manera de tiempo se puede transferir de una máquina a otra por medio de mensajes de forma que pueda ser comparada con otra marca de tiempo generado localmente. El problema que se plantea es el de sincronizar los ordenadores, para ello se puede mandar un mensaje por medio de la red de comunicaciones, el problema es medir el tiempo que se tarda en enviar el mensaje. Este tiempo está limitado por el tiempo de transferencia de la red, pero su valor real es impredecible, por lo que es muy difícil diseñar protocolos que sincronicen los ordenadores. En SD muchas veces basta con controlar el oren de los procesos (ej. actualización del fichero).

Consistencia de interfaz de usuario

Se produce cuando en un programa interactivo se pulsa una tecla o el ratón y la pantalla permanecen inconsistentes con la acción que se realiza. El programa procesa la entrada al usuario y realiza las acciones necesarias para controlar la pantalla. El usuario se percatará de la inconsistencia de la pantalla salvo que se procesen los cambios rápidamente y el usuario lo percate como algo instantáneo. Para conseguir esta actividad se deben obtener veinte imágenes por segundo, es decir, el retardo máximo debe ser 0'15. Este retardo se calcula como el tiempo necesario para recibir y procesar la entrada y calcular los cambios a la ventana y modificar la imagen mostrada. Estos tiempos son importantes en aplicaciones altamente interactivas o de imágenes gráficas.

Sistemas Distribuidos

1

2