Sistemas Distribuidos.

1. Conceptos fundamentales.

Definición de un sistema de cómputo distribuido.

La idea fundamental de un sistema distribuido es que constituye una combinación de computadoras y sistemas de transmisión de mensajes bajo un solo punto de vista lógico a través del cual los elementos de cómputo resuelven tareas en forma colaborativa. Se puede aseverar que el sistema constituye un ente capaz de procesar información debido a dos características esenciales:

- El sistema consiste de una cantidad de computadoras cada una de las cuales tiene su propio almacenamiento, dispositivos periféricos y potencia computacional.
- Todas las computadoras están adecuadamente interconectadas.

Por medio del sistema operativo adecuado, las computadoras mantienen su capacidad de procesamiento de tareas local, mientras constituyen elementos colaborativos de procesamiento en el ambiente distribuido. El elemento de interconexión indica que debe existir el mecanismo de transporte de información entre los componentes de manera que sea factible el intercambio de mensajes entre nodos cooperativos de manera que no se violente la transparencia de una transacción. Desde el punto de vista del usuario, un Sistema Operativo Distribuido se comporta como un sistema operativo convencional que se ejecuta en su computadora local; sin embargo, éste administra los recursos de varias –y además, posiblemente heterogéneas- computadoras independientes e integra una interface común hacia el usuario. Se puede decir entonces que un ambiente distribuido también incluye las siguientes características:

- Una variedad de componentes que incluyen tanto plataformas de cómputo como las redes de interconexión que transportan mensajes entre ellas unificadas en un solo ambiente de procesamiento.
- La transparencia, como resultado de la abstracción apropiada de los componentes del sistema.

Con respecto a la transparencia, no siempre es posible mantenerla estrictamente en todos los casos. Algunas ocasiones es necesario el establecimiento de ciertas distinciones entre los usuarios y los operadores del ambiente computacional, debido a las tareas específicas o generales que éstos deben realizar. La transparencia implica requerimientos de integración de los componentes en una sola unidad:

- Un SD es un sistema operativo de nivel amplio.
- Un SD provee componentes abstractos del sistema y en muchos casos está basado en ellos.
- Un SD implementa control distribuido de acuerdo al principio de la autonomía cooperativa.
- Existe una descripción distribuida del sistema.

Un SD es un sistema operativo de nivel amplio.
Un SD permite la cooperación entre diferentes máquinas o elementos de procesamiento que están conectados al mismo medio de transporte. La transparencia exige que el uso de los componentes sea independiente del tipo y la localidad donde sean empleados en un momento dado.

Un SD provee componentes abstractos del sistema y en muchos casos está basado en ellos.
Su provisión y uso se puede apreciar como el manejo de características abstractas por medio de las cuales se logra emplear dichos recursos. Otro enfoque diferente del anterior es el desarrollar un SD a partir de controladores (drivers) especiales para las máquinas que lo integrarán. En ambos casos el diseño resulta en una interface en extremo cercana al hardware que soporta la operación del sistema, pero representado para el usuario de una forma coherente y lógica.

Un SD implementa control distribuido de acuerdo al principio de la autonomía cooperativa.
El principio de autonomía cooperativa establece que todos los elementos involucrados tienen igual oportunidad en la toma de decisiones que les afectan como un todo. En un momento dado, la toma de decisión se hace sólo entre los procesadores involucrados en una tarea, en tanto que los demás permanecen ajenos a esta decisión. Visto desde la óptica individual, los procesos de decisión autónomos cooperativos exhiben una mayor complejidad que aquellos basados en decisiones jerárquicas.

Existe una descripción distribuida del sistema.
La consistencia semántica significa que todos los servicios del sistema, los directorios y programas de aplicación frecuentemente usados deben tener el mismo efecto sin importar el punto donde sean ejecutados, y ser llamados de la misma forma independientemente de la ubicación y notación interna del nodo. Desafortunadamente, el objetivo de consistencia semántica entra en conflicto con los objetivos de autonomía local, de manera que restringe en forma importante las posibilidades de instalar software en los nodos, así como la facilidad de adaptar dicho software al usuario.

Existen ciertos factores que han propiciado el auge tan elevado de los sistemas distribuidos dentro del procesamiento de las organizaciones modernas en el mundo, en particular Tanenbaum [1] señala algunos de ellos como:

1. Avances en tecnología de cómputo.
2. Desarrollo de las redes locales de alta velocidad (LAN).
3. Desarrollo de redes de área amplia (WAN).

A los que se podrían agregar: la creación y proliferación de las redes de interconexión, que permiten el acoplamiento de sistemas de procesamiento y periféricos, así como la aparición de software con una mejor ingeniería, lo cual permite el escalamiento y actualización hacia nuevas capacidades y prestaciones de acuerdo a las necesidades de desempeño de las aplicaciones actuales. Muchos autores hacen la distinción entre los *sistemas distribuidos*, elaborados para que trabajen con ellos varios usuarios en forma simultánea y los *sistemas paralelos*, que pretenden lograr la máxima velocidad de ejecución en una tarea

determinada. Esta distinción ha tenido mucha controversia, por lo que el consenso general ha determinado usar "sistema distribuido" en sentido amplio, donde varios CPU´s trabajan –intercomunicados entre sí- de manera cooperativa.

Los sistemas distribuidos plantean importantes ventajas en comparación con los sistemas tradicionales centralizados, entre ellos que muchas aplicaciones están elaboradas para operar de forma natural en ambiente disperso tales como: bases de datos, sistemas de trabajo cooperativo (como *Madefast*) y juegos cooperativos o MUD´s (se puede apreciar un ejemplo de esto usando el software de demostración en tiempo real que se puede obtener de http://www.activeworlds.com). Otra ventaja importante la constituye una mayor confiabilidad, pues es factible construir un sistema tolerante a fallas si sus componentes comparten la carga de trabajo y, de presentarse alguna falla, dicha carga es asignada a los demás elementos que siguen operando.

Por lo que respecta a la comparación de un sistema distribuido contra la operación de una sola PC, las dos ventajas esenciales son la compartición de periféricos/recursos y la distribución de carga de las aplicaciones. En el primer caso, se logra tener acceso a equipos sofisticados o especializados que al ser usados concurrentemente por varias personas permiten aplicar criterios de economía de escala para su asignación y adquisición desde el punto de vista económico. En cuanto al segundo punto, a través de un ambiente distribuido se logra mejorar la carga de trabajo de ciertos equipos o servicios, de manera tal que se balancea adecuadamente el nivel de desempeño y aplicación de componentes de hardware y software, maximizando el uso de los sistemas de acuerdo a la demanda de los usuarios.

Sin embargo, frente a todas las ventajas de los SD´s, existen desventajas importantes que deberán ser resueltas de la mejor manera posible antes de considerar un estado maduro de tales sistemas en lo que respecta al uso difundido de ellos dentro de las aplicaciones cotidianas de las organizaciones modernas. Los tres principales señalados también por Tanenbaum [1] son el software, las redes de interconexión y la seguridad. El software se considera una problemática importante dado que para un SD es importante tomar en cuenta que éste debe ser de un tipo y capacidades muy especiales, hecho específicamente para administrar y operar sobre un ambiente disperso. En este rubro, la mayor parte de los sistemas desarrollados están pensados para operar sobre una computadora ya sea centralizada o bien, monousuario, dejando de lados las problemáticas de sincronización, control y distribución de carga que aparecen en los ambientes distribuidos. Las redes de interconexión son relevantes porque a través de ellas fluyen los mensajes y paquetes de comunicación entre los diferentes procesadores involucrados en la tarea, y de fallar éstas los procesos asociados pueden dañarse o interrumpirse. Finalmente, la cuestión de la seguridad aparece también como un elemento a considerar, dado que si cualquier usuario tiene acceso a través de una imagen lógica a todos los elementos y periféricos que integran un SD, también puede leer información o datos de otras personas (por lo menos en forma potencial).

Es un error común de enfoque el suponer iguales un sistema operativo de red y un sistema operativo distribuido, el cual anima y maneja los recursos del sistema distribuido en sí. Existen diferencias importantes en la operación interna de tales sistemas, que enseguida se mencionan:

- En una red, cada computadora ejecuta su propio sistema operativo, y no como parte de un sistema operativo general.
- Cualquier actividad en computadoras remotas (como servidores, por ejemplo) se lleva a cabo por medio de accesos (login) remotos en dichas computadoras que se hacen en forma explícita por parte del usuario, y no como una función de los procesos como ocurre en un sistema distribuido.
- El trabajo con archivos remotos, igual presupone transferencias de archivos explícitas donde el usuario especifica la localización remota y no se da esta asignada por el ambiente operativo.
- Las facultades de tolerancia a falla son un poco más pobres, pues cuando una computadora falla, esto no influye grandemente en la degradación del servicio.

Las funciones más importantes de un Sistema Distribuido.

Un SD posee las siguientes funciones básicas:

- Comunicación interprocesos. La intención es tener comunicación entre procesos sobre la red de conexión, usando para ello las facilidades instaladas. Generalmente implica el uso de un protocolo de transporte para establecer el enlace en el ámbito de protocolo de servicios.
- Administración y asignación de recursos. Contempla la asignación de recursos a usuarios, toma de decisión de en dónde deberán ser ejecutadas las peticiones, creación e instalación de nuevos recursos en la red, soporte de replicación para procesos críticos, mecanismos de control de concurrencia y sincronización.
- Administración de nombres. Mecanismos de asignación y mantenimiento de los nombres de los recursos, localización de servidores/usuarios, mantenimiento de directorios, etc.
- Reinicio luego de fallas. Implementado en varias capas de operación a lo alto de la arquitectura del sistema.
- Funciones de protección. Especificación de los usuarios y sus derechos de acceso, mecanismos de autentificación, políticas de acceso y contra ataques externos.

Desde el punto de vista del usuario, algunos de los requisitos pueden ser:

- Proveer de ayudas para la solución de problemas
- Minimización del costo de acceso a los recursos
- Maximización y simplificación de las facilidades de comunicación con otros usuarios o programas.

Evolución de los sistemas de cómputo distribuido.

Las primeras computadoras eran dispositivos de muy elevado costo, que existían casi exclusivamente en las grandes universidades y centros de investigación. Estas computadoras iniciales controlaban su ejecución desde una consola no accesible a los usuarios, quienes entregaban su trabajo a los operadores generalmente en forma de tarjetas perforadas a fin de que pudieran correr los programas y entregar los resultados más tarde. La especificación del ambiente para ejecución de trabajos en particular (cintas, lectoras de tarjetas, etc.) desperdiciaba mucho del valioso tiempo de procesador. Varios conceptos nuevos se introdujeron en las décadas de los 50´s y 60´s para optimizar el tiempo de procesamiento y el acceso/configuración de los recursos necesarios para llevara a cabo un programa o grupo de programas en particular. Entre esas nuevas metodologías de operación destacan: el trabajo por lotes (donde se agrupaban programas que demandaban recursos similares y se ejecutaban en serie uno tras otro), secuenciación automática de trabajos, procesamiento fuera de línea a través de los conceptos de *buffering* (memorias intermedias de alta velocidad para optimizar la entrada-salida de datos), *spooling* (almacenamiento en colas de espera de los trabajos de impresión), y finalmente la aparición de los entornos de multiprogramación.

Sin embargo, ninguna de estas innovaciones permitían que múltiples usuarios interactuaran simultáneamente con el sistema, como cuando era necesario depurar programas con errores. Al inicio de la década de los 70´s las computadoras comenzaron a emplear las rutinas de tiempo compartido, las cuales a través de dispositivos -conocidos como terminales tontas- conectados directamente a la computadora principal permitían ahora ejecutar tareas interactivas en forma simultánea y compartir los recursos del sistema de cómputo. Al mismo tiempo se tuvieron avances importantes en tecnología de electrónica que permitieron la reducción de tamaño de los grandes equipos centralizados, mientras se incrementaba la capacidad y velocidad de procesamiento y se disminuían sensiblemente sus costos. Los nuevos sistemas fueron conocidos como *minicomputadoras*. Estas dos ideas principales –tener un equipo conectado a la computadora principal a través del cual se pudiera hacer uso de sus recursos y la flexibilidad de acceso sin importar la localización física- fueron retomadas para el desarrollo ulterior de los sistemas distribuidos.

Durante la década de los 70´s aparecieron las primeras estaciones "inteligentes" para usuarios, es decir equipos con capacidad de procesamiento y memoria local, de forma tal que ahora se conectaban al equipo central sistemas con capacidad de ejecución local. Esto marcó la aparición de las primeras computadoras de alto rendimiento conocidas como *workstations*, las cuales sin embargo todavía estaban atadas a las restricciones de cercanía con el equipo central, dado que se usaban cables de interconexión cuyas longitudes no podían exceder de ciertos rangos de tolerancia para que la comunicación entre procesadores se pudiera llevar a cabo confiablemente.

Por otra parte y en conjunto con estos avances, se estaban dando los primeros pasos en la creación de sistemas de comunicación que luego emergieron como tecnologías de redes –LAN, *local area network* y WAN, wide area network- los cuales permitieron que varias computadoras localizadas dentro de un edificio o campus pudieran intercambiar información en tasas muy altas de velocidad. Los sistemas LAN permitían típicamente

transmisión a 10 Mbps, mientras que los WAN soportaron equipos separados inclusive entre ciudades o países a velocidades que iniciaban en 56 Kbps. La primera red local de amplio impacto comercial se llamó Ethernet y fue inventada por Xerox en 1973, mientras que la primera WAN fue conocida como ARPANET (advanced research projects agency network), desarrollada por el departamento de la defensa de Estados Unidos en 1969.

A lo largo de los 80´s las velocidades de las redes se incrementaron, con alcances de hasta 100 Mbps para las LAN y de 2.048 Mbps para WAN. A inicio de esta década de los 90´s ha aparecido la tecnología ATM que permitirá la transmisión de información hasta en el rango de los 1.2 Gbps tanto para ambientes locales como de área amplia. La disponibilidad de estas redes de banda ancha impulsará la creación aplicaciones futuras para cómputo distribuido que soportarán una nueva generación de aplicaciones dispersas, llamadas aplicaciones multimedia. Estas permitirán el manejo de diversas fuentes de información simultáneamente como datos, voz, video, telefonía, etc., las cuales extenderán los alcances funcionales que se han usado hasta el momento.

La mezcla de las tecnologías de cómputo y comunicaciones durante finales de los 70´s e inicios de los 80´s permitieron el nacimiento de una nueva generación de aplicaciones que fueron los sistemas distribuidos. Dado que el hardware de comunicaciones y de procesamiento de datos estaba bastante maduro y dominado al inicio de los 70´s, los mayores esfuerzos se enfocaron en la generación de software que pudiera mantener operando estos sistemas y que resultara en una nueva generación de sistemas para las empresas. Se lograron bastantes avances en las universidades respecto a este campo, y actualmente han aparecido ya los primeros sistemas operativos distribuidos en forma comercial.

Sistemas Distribuidos
Ejercicio # 1-1

1. Instale en su computadora el software de demostración de http://www.activeworlds.com ,donde podrá hacer uso de una aplicación distribuida tipo MUD. Explique cómo se aprecia en la funcionalidad del ambiente las siguientes características de un sistema distribuido:

- Un SD es un ambiente de nivel amplio.
- Un SD provee componentes abstractos del sistema y en muchos casos está basado en ellos.
- Un SD implementa control distribuido de acuerdo al principio de la autonomía cooperativa.
- Existe una descripción distribuida del sistema.

2. Lea la justificación técnica del proyecto seti@home en http://setiathome.ssl.berkeley.edu/about_seti/about_seti_at_home_1.html . De acuerdo a lo explicado ahora en el curso, justifique ampliamente si este puede considerarse un ejemplo de computación distribuida y por qué. Instale en su computadora el software de análisis de señales radioeléctricas del proyecto SETI del sitio Web http://setiathome.ssl.berkeley.edu/windows.html para comprobar el funcionamiento de un sistema que envía información a equipos remotos y luego recupera la salida general de todos ellos.

3. Investigue en Internet cómo opera el sistema ***Madefast*** y discuta los mecanismos que implementa como ambiente distribuido de trabajo.

4. Mencione las diferencias entre los siguientes tipos de sistemas operativos mediante la definición de sus propiedades esenciales.

a) Un sistema de tiempo compartido.
b) Procesamiento paralelo.
c) Sistema de Red.
d) Sistema Distribuido.

5. Establezca con claridad cuáles son las diferencias fundamentales entre un sistema operativo de red y un sistema operativo distribuido.

Modelos de computación distribuida.

Existen varios modelos fundamentales que se usan para la creación de sistemas de cómputo distribuido, los cuales se clasifican en diferentes categorías –modelo de minicomputadora, modelo de workstation, modelo workstation-server, modelo de pool de procesadores e híbrido. Enseguida se presentan brevemente, junto con las notas distintivas esenciales.

Modelo de minicomputadora

Es simplemente la extensión del modelo de equipo centralizado. Este esquema consiste de varias minicomputadoras conectadas por una red de comunicación. Cada minicomputadora tiene su propio grupo de usuarios quienes pueden tener acceso a otros recursos no presente en su sistema a través de la red de datos. Este modelo se usa cuando existe necesidad de compartir recursos de diferentes tipos a través de acceso remoto. Ejemplo: los inicios del ARPANet.

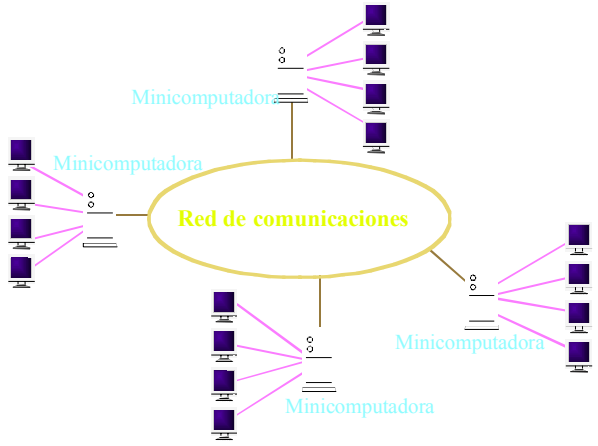

Fig. ***1-1***. Modelo de minicomputadora.

Modelo de workstation

Consiste en varias workstations interconectadas por una red, como lo muestra la Fig. 1-2, cada usuario se "da de alta" en su computadora de inicio o "computadora *home*" y luego desde ahí puede enviar trabajos para ejecución. Cada estación tiene su propio disco duro, y su propio sistema de archivos. La implementación mostrada tiene la desventaja de que se puede desperdiciar tiempo de procesamiento si una o más computadoras no están haciendo uso de su CPU. Si el sistema determina que la estación del usuario no posee la suficiente capacidad de proceso para una tarea, transfiere el trabajo de forma automática hacia el equipo que tiene menos actividad en ese momento y por último se regresa la salida del trabajo a la estación del usuario. Ejemplo: el sistema Sprite desarrollado experimentalmente por Xerox.

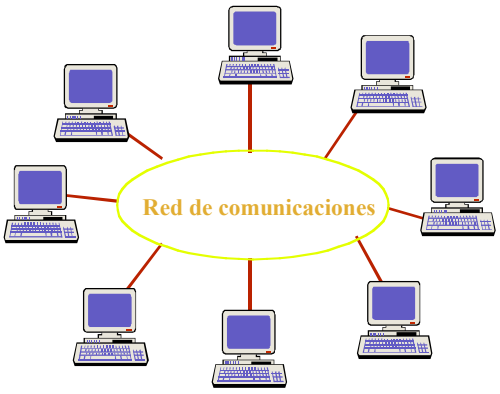

Fig. ***1-2***. Modelo de Workstation.

Para la implantación del modelo en la vida real hay que resolver ciertos problemas:

1. ¿Cómo identifica el sistema a una estación desocupada?
2. ¿Cómo puede transferirse un proceso desde una estación para ser ejecutado en otra?
3. ¿Qué le ocurre a un proceso remoto si un usuario se da de alta en una estación que estaba desocupada hasta ese momento y estaba siendo utilizada para correr un proceso de otro usuario?

Más adelante se presentarán los diferentes enfoques para resolver esta problemática.

Modelo workstation-server

Este se ilustra en la Fig. 1-3. Posee varias microcomputadoras y varias estaciones de trabajo (algunas de las cuales pueden ser estaciones sin disco duro, o *diskless*) comunicadas mediante red. Ahora que existe la potencialidad de tener estaciones sin disco, el sistema de archivos a usar por estas computadoras puede ser el de una computadora completa o alguno compartido de las diferentes minicomputadoras del sistema. Algunas otras microcomputadoras se pueden usar para proveer otros tipos de servicios, como base de datos, impresión, etc. Estas máquinas cumplen ahora nuevas tareas especializadas para proveer y administrar acceso a los recursos, en la jerga común se les llama *servidores* (servers).

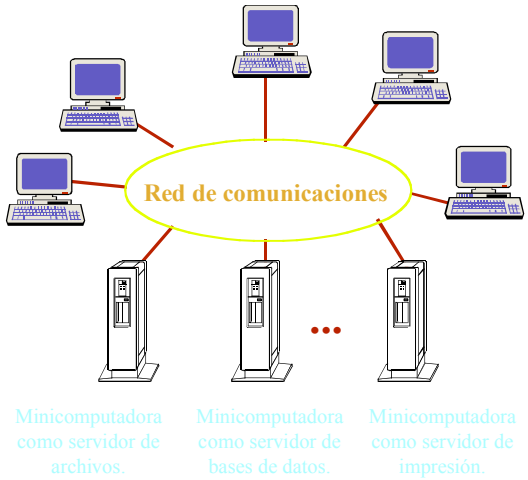

Fig. ***1-3***. Modelo Workstation-Server.

Por razones de escalabilidad y confiabilidad, es común distribuir un recurso entre varios servidores, lo cual incrementa la confiabilidad y rapidez en la solución de fallas en caso de algún mal funcionamiento de alguno de ellos. Por ejemplo, el servicio de correo electrónico puede tenerse en diferentes equipos del conjunto, de manera tal que si uno fallara momentáneamente, fuese posible para todos los demás usuarios continuar trabajando con dicho servicio a través de los equipos redundantes.

En este modelo el usuario se da de alta normalmente en su computadora y para ciertas tareas utiliza los recursos locales. Cuando necesita otro servicio que no tiene instalado, las peticiones de aplicaciones específicas se encaminan hacia los servidores, quienes procesan las solicitudes y regresan al usuario el archivo o programa solicitado. Otra característica es que en este modelo los procesos del usuario no necesitan ser migrados hacia los servidores para obtener los beneficios del procesamiento adecuado. Como observaciones al modelo, se tienen las siguientes:

1. En general es más barato usar unas pocas minicomputadoras equipadas con grandes discos de alta velocidad que tener a todas las workstations con un disco duro instalado más lento y más pequeño.
2. Desde el punto de vista de mantenimiento, es mejor tener estaciones sin disco que todo el sistema completo.
3. El usuario tiene en general la libertad de elegir en cual estación trabajar, a diferencia del modelo de workstation, donde tiene su cuenta y clave de acceso.
4. Su operación se basa en el paradigma cliente-servidor, donde un proceso cliente envía una petición de procesamiento al server. El servidor ejecuta la petición y regresa a la estación el resultado de la misma.

Ejemplo: el ambiente V-System.

Modelo de pool de procesadores

Este se basa en el hecho de que los usuarios en promedio no requieren capacidad de procesamiento durante un buen rato, pero existen instantes en los que la actividad y los programas que ejecutan demandan potencia de trabajo en alto grado. A diferencia del modelo anterior en el que cada persona tiene su servidor asignado, en éste se dispone de un conjunto (pool) de servers que son compartidos y asignados conforme a demanda. Cada procesador en el pool tiene su propia memoria y ejecuta un programa de sistema o de aplicación que le permite participar en el ambiente de cómputo distribuido.

Se puede apreciar en la Fig. 1-4, que el modelo no soporta la conexión de estaciones directamente a los servidores, sino sólo por medio de la red de interconexión. Las terminales usadas pueden ser estaciones sin disco o terminales gráficas como *X terminals*. Se tiene instalado un servidor especial llamado "de ejecución" el cual se dedica a la administración y asignación de recursos conforme a la demanda. Cuando se recibe una solicitud de una persona, este equipo asigna temporalmente el número de servers que deberán ser dedicados al trabajo de tal petición, y luego de atenderla regresan para ser liberados y quedar a disposición de la siguiente tarea. Otra diferencia importante con otros esquemas, es que aquí no existe el concepto de "computadora home", de manera que el usuario no se da de alta en alguna máquina en particular, pero sí percibe al sistema como un todo.

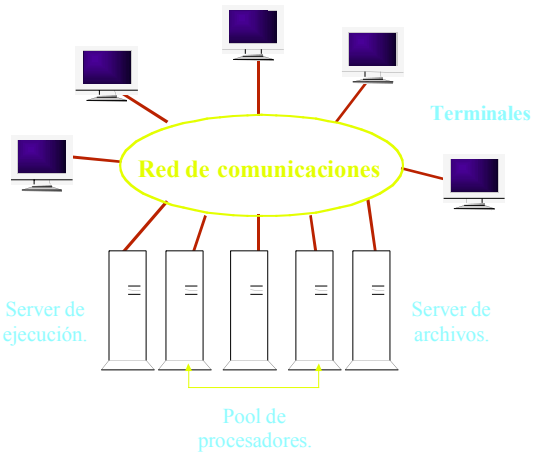

*Fig. **1-4***. Modelo *pool* de servers.

Comparado con el modelo de Workstation-Server, el de Pool de Servers permite una mejor utilización del poder de cómputo disponible en el ambiente distribuido, dado que dicho poder computacional esta disponible para todos a diferencia de Workstation-Server, donde varios equipos pueden estar desocupados en algún momento pero su capacidad no se puede asignar a otros. Por otra parte, esta metodología provee gran flexibilidad ya que el sistema se puede expandir agregando procesadores que operarán como servidores adicionales para soportar una carga extra de trabajo originada por incremento en el número de usuarios o por la implantación de nuevos servicios.

Sin embargo, este modelo no se considera recomendado para programas de cálculo complejo como aplicaciones intensivas de alto rendimiento o aplicaciones basadas en gráficas. Esto se debe principalmente a la lentitud de las redes de conexión que se utilizan para la comunicación entre procesos. Para este tipo de tareas se considera mejor el modelo Workstation-Server.

Ejemplos: Amoeba, Plan 9 y el Cambridge Distributed Computing System.

Modelo híbrido

De los cuatro modelos descritos anteriormente, el de Workstation-Server es el más ampliamente utilizado para la construcción de sistemas de cómputo distribuido. Esto se debe a que un usuario promedio solamente ejecuta tareas interactivas simples como editar archivos, enviar correos electrónicos, etc. El modelo se ajusta perfectamente a especificaciones tan sencillas. Para ambientes donde hay grupos de usuarios de mayor potencia que realizan tareas masivas con alta necesidad de poder computacional el modelo Pool de Servers es más adecuado.

El modelo híbrido permite combinar las mejores características de Workstation-Server y Pool, esencialmente agregando a la red de estaciones de trabajo un Pool de servidores que pueden ser asignados dinámicamente para trabajos que son muy extensos para máquinas individuales o que necesitan varios equipos concurrentemente para una ejecución adecuada. Esta variante tiene la ventaja de garantizar el nivel de respuesta a trabajos interactivos dado que permite la ejecución en la misma computadora de la persona que lo solicita. Por otra parte, su principal desventaja estriba en que el costo de implantación se eleva puesto que requiere mayor número de componentes para su construcción.

Taxonomía de los ambientes de cómputo.

No existe una clasificación general que sea totalmente aceptada para poder ordenar los ambientes de cómputo existentes y dentro de ellos, a los sistemas distribuidos en particular. La clasificación de *Flynn* (1966) está fundamentada en la multiplicidad de los flujos de instrucciones y de los flujos de datos dentro del sistema. El esquema de *Feng* (1972) se basa en la confrontación entre procesamiento serie frente a procesamiento paralelo. La clasificación de *Händler* (1977) se basa en el grado de paralelismo y encauzamiento de los diferentes subsistemas.

En la taxonomía propuesta por Flynn, las computadoras digitales pueden clasificarse en cuatro categorías con respecto a sus flujos de instrucciones y datos. El proceso de computación esencial es la ejecución de una secuencia de instrucciones sobre un flujo de datos. El término *flujo* se emplea para denotar una secuencia de elementos (que pueden ser instrucciones o datos) que ejecuta un solo procesador. Las instrucciones o los datos se definen con respecto a la máquina referenciada. Un *flujo de instrucciones* es una secuencia de instrucciones ejecutadas por la máquina; un *flujo de datos* es una secuencia de datos que incluye los datos de entrada y los resultados parciales o totales solicitados o producidos por el flujo de instrucciones.

De esta manera, se tiene que las diferentes organizaciones de computadoras se caracterizan por la multiplicidad de hardware provisto para atender a los flujos de instrucciones y de datos. La categorización de Flynn depende de la multiplicidad de procesos simultáneos que ocurren en los componentes del sistema, aunque conceptualmente –como veremos- sólo son necesarios tres tipos de equipos de dicha taxonomía.

Las instrucciones y datos se toman de los *módulos de memoria*. Las instrucciones se decodifican en la *unidad de control*, que envía el flujo de instrucciones decodificadas a las *unidades procesadoras* para su ejecución. Los flujos de datos circulan entre los procesadores y la memoria bidireccionalmente. Se pueden utilizar múltiples módulos de memoria en el subsistema de memoria compartida donde cada flujo de instrucciones es generado por una unidad de control independiente. Los múltiples flujos de datos se originan en el subsistema de módulos de memoria compartida. Por simplicidad no se mostrarán los puertos de E/S dentro de los diagramas.

La taxonomía propuesta por Flynn es como sigue.

SISD: Single Instruction, Single Data.
Sistemas enfocados a la operación con un solo procesador, que representan a la mayoría de las computadoras disponibles actualmente. Las instrucciones se ejecutan secuencialmente, pero pueden estar traslapadas en las etapas de ejecución (lo que se conoce como segmentación encauzada). Es importante mencionar que un equipo SISD puede tener más de una unidad funcional, pero todas las unidades funcionales están bajo la supervisión de una sola unidad de control. En la siguiente Fig. 1-5 aparece un esquema de este tipo de sistemas (FD= flujo de datos, FI= flujo de instrucciones, UC= unidad de control, UP= unidad de procesamiento, MM= módulo de memoria).

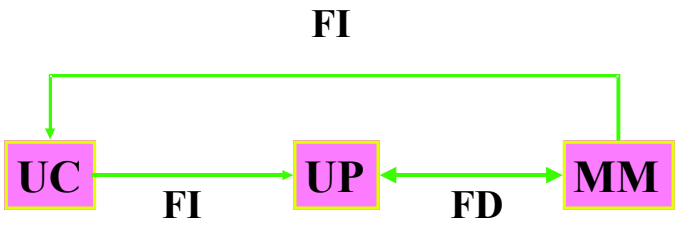

Fig. ***1-5***. Sistema SISD.

SIMD: Single Instruction, Multiple Data.
En estos sistemas existen múltiples elementos de procesos supervisados por una misma unidad de control. Todos los elementos de proceso reciben la misma unidad de instrucciones emitida por la unidad de control pero operan sobre diferentes conjuntos de datos procedentes de flujos distintos. El subsistema de memoria compartida puede contener múltiples módulos. Se tiene entonces que la unidad de control selecciona una instrucción y luego instruye a varias unidades de procesamiento para la ejecuten, cada una con sus propios datos. Estas máquinas se pueden dividir también en: sección de palabra y sección de bit.

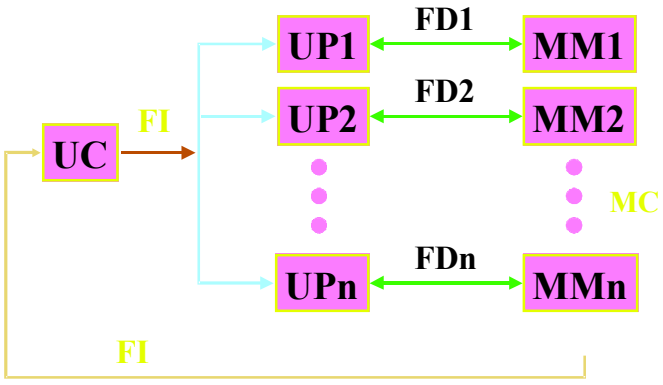

Fig. ***1-6***. Sistema SIMD.

MISD: Multiple Instructions, Single Data.
Para estas máquinas existen *n* unidades procesadoras, cada una recibe distintas instrucciones que operan sobre el mismo flujo de datos y sus derivados. Los resultados de un procesador se pasan como entrada del siguiente dentro del cauce de procesamiento. En este caso habría varias instrucciones a procesar, con un único flujo de datos. Ninguna computadora conocida se ajusta a esta descripción, dado que ésta arquitectura se considera poco práctica.

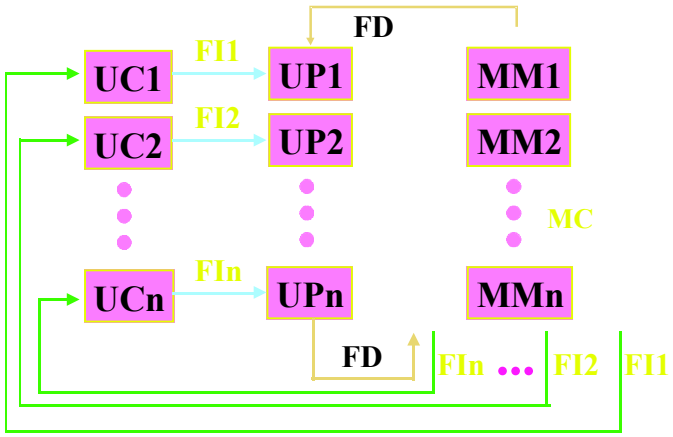

*Fig. **1-7***. Sistema MISD.

MIMD: Multiple Instructions, Multiple Data.
La mayoría de los sistemas multiprocesadores y de los sistemas con múltiples computadoras pueden incluirse en esta categoría. Este se integra con múltiples computadoras independientes, cada una con su propio contador de programa y datos. Un equipo MIMD intrínseco implica interacciones entre los *n* procesadores porque todos los flujos de memoria se derivan del mismo espacio de datos compartido por todos los procesadores. Si los *n* flujos de datos provinieran de subespacios disjuntos dentro de las memorias compartidas, entonces tendríamos la llamada operación *SISD múltiple* (MSISD), que no es más que una conjunto de *n* sistemas monoprocesadores SISD independiente. Un sistema MIMD intrínseco es *fuertemente acoplado* si el grado de interacciones y conectividad física entre los procesadores es elevado, en caso contrario los consideramos *débilmente acoplados*. La mayoría de los sistemas MIMD comerciales son débilmente acoplados.

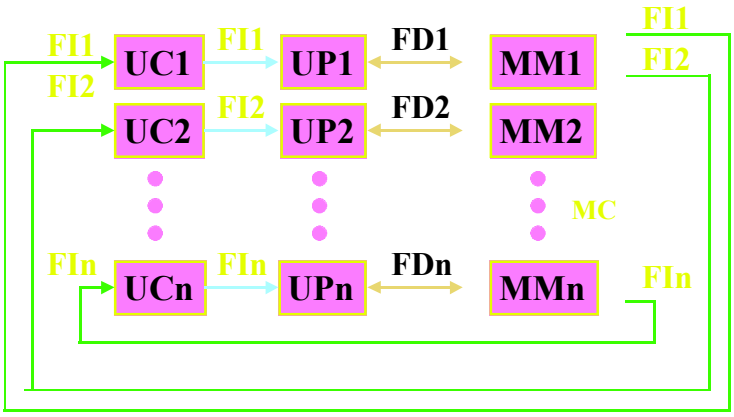

*Fig. **1-8***. Sistema MIMD.

La clasificación de los equipos MIMD se puede ampliar señalando que las computadoras que tienen memoria compartida muchas veces son llamadas **multiprocesadores**, las que no

siguen este esquema y mantienen la memoria privada para cada máquina son las **multicomputadoras**. Cada una de estas categorías a su vez se puede dividir basándose en su arquitectura de conexión, ya sea siguiendo el esquema de bus de comunicaciones (red, backplane o cable) que forma un medio de difusión entre ellos; o mediante conmutador (switch) en una topología que realmente tiene varios enlaces de comunicación compartidos entre los equipos que pueden encaminar mensajes etapa por etapa gracias a un sistema de conmutación que dirige el mensaje a lo largo de los cables de salida.

La taxonomía de Feng se basa en el grado de paralelismo como norma de clasificación de las arquitecturas, donde se entiende como *máximo grado de paralelismo P* al número máximo de dígitos binarios (bits) que pueden ser procesados en una unidad de tiempo por la computadora. Si P_i es el número de bits que pueden ser procesados en el i-ésimo ciclo de procesador (o en el i-ésimo período de reloj), y si se consideran T ciclos de procesador indicados por i= 1, 2, ... , T, el grado medio de paralelismo P_m viene definido por:

$$Pm = \frac{\sum_{i=1}^{T} Pi}{T} \qquad (1.0)$$

En general $P_i < P$, por lo tanto, está la tasa de utilización μ de un sistema de cómputo durante T ciclos como:

$$\mu = \frac{Pm}{P} = \frac{\sum_{i=1}^{T} Pi}{TP} \qquad (1.1)$$

Si la potencia de la computadora está totalmente utilizada (es decir, si el paralelismo está totalmente aprovechado), tenemos entonces que $P_i = P$ para todo i, con lo que $\mu = 1$ con utilización del cien por ciento. La tasa de utilización depende del programa de aplicación que se ejecute.

Otra forma de poder visualizar el grado de paralelismo de un sistema es considerar su longitud de palabra n y la longitud m de sección de bits. La sección de bits es una cadena de bits, formada por cada una de las palabras sobre las que se procesa en paralelo. El máximo grado de paralelismo se expresa como el producto de n por m, es decir:

$$\mathbf{P = nm} \qquad (1.2)$$

Ejemplo: calcule el grado de paralelismo para una computadora con longitud de palabra de 64 bits y que tiene cuatro cauces aritméticos, cada uno de 8 etapas.

R: la sección de bits mide 8x4 = 32 bits, por lo tanto P = 62 x 32 = 2048.

Precisamente, el par ordenado (*n*, *m*) corresponde a los puntos en el espacio de sistemas de cómputo para el sistema de coordenadas de la Fig. 1-9, donde se muestra la esta clasificación de acuerdo a como lo presenta Hwang [7].

También se puede ampliar el concepto si para establecer con mayor claridad el grado de paralelismo se presentan las siguientes definiciones:

Tiempo de respuesta = el tiempo promedio necesario para completar una tarea (lo nombraremos R: en segundos/tarea).

Rendimiento máximo (o throughput) = el número promedio de tareas que pueden ser terminadas en una unidad de tiempo (nombrado T: en tareas/segundo).

Establecidas estas definiciones, se puede tener otro enfoque del grado de paralelismo (D) como sigue:

$$D = R * T \qquad (1.3)$$

Ejemplo: si en un sistema distribuido se completan 20 tareas cada 10 segundos, y cada tarea se ejecuta en 3 segundos, ¿Cuál es su grado de paralelismo? ¿Qué significa el resultado o cómo puede interpretarse?

R: el rendimiento es de 20/10 tareas/seg, y el tiempo de respuesta es de 3 segs. Entonces, el grado de paralelismo es D = 20/10 * 3 = 6. En otras palabras, se ejecutan un promedio de 6 tareas en paralelo dentro del sistema.

Pipelining y paralelismo.

Es posible decrementar el tiempo necesario para completar una tarea dentro de una computadora dividiéndola en N subtareas independientes. Así, N procesadores pueden ejecutar las diferentes subtareas. Más aún, es factible lograr un incremento lineal en la velocidad si la división en N subtareas se puede hacer de tal modo que el rendimiento máximo incrementado se incremente en un factor de N. Por el contrario, una descomposición no-óptima reducirá el incremento en velocidad a un factor menor que N.

Ejemplo: una página en el WWW contiene varias imágenes. Un navegador web puede descargar todas las imágenes una por una o puede tratar de leerlas todas en paralelo. Suponiendo que se puede asignar un procesador para descargar cada imagen. Cada descarga es independiente de la otra, de manera que el tiempo de terminación para la versión en paralelo es el tiempo que tarda en completarse la operación más lenta. Si un sitio tiene tres imágenes, con tiempos de descarga de 3, 5 y 10 segundos, la versión serial no paralelizada se demora 18 segundos en concluir, en tanto que a la paralela le lleva 10 segundos. La razón de incremento es de 18/10 = 1.8, que está determinada por el tiempo más lento para leer una imagen. Si a cada imagen le lleva el mismo tiempo descargarse, es decir 6 segundos, entonces la razón de incremento sería de 18/6 = 3. Esta es la descomposición optima de la tarea.

Si es posible dividir una tarea T en las subtareas T_1, T_2, ... T_N, de manera que T_k puede comenzarse luego de que terminó de ejecutarse T_{k-1}, se dice que las tareas son serialmente dependientes. Más aún, si es posible dividir una tarea en N subtareas serialmente dependientes, entonces un pipeline de N procesadores puede dar un rendimiento N veces más grande que un solo procesador.

Ejemplo: se deseó paralelizar un stack de protocolos asignando un procesador a la capa de enlace, uno a la de red, otro a la de transporte y finalmente uno a la de aplicación. Asumiendo que cada capa consume la misma cantidad de tiempo –por decir algo, 10 ms- para procesar un paquete (una suposición totalmente falaz). En la versión no paralela (serial) del stack, el sistema requiere 40 ms para procesar un paquete, dado que no puede admitir otro nuevo hasta que termina con el actual. En la versión con pipeline, cada paquete tiene un retardo de 40 ms, como en la versión serial, pero el pipeline procesa un paquete cada 10 ms, en lugar de un paquete cada 40 ms.

Sin embargo, el cálculo del grado de paralelismo se puede complicar si las etapas del pipeline requieren diferentes tiempos para concluir su procesamiento, en cuyo caso se puede emplear la ecuación (1.3) con las siguientes modificaciones:

Tiempo de respuesta = el tiempo necesario para completar las tareas en las diferentes etapas (lo nombraremos R: en segundos/tareas).

S = el tiempo que necesita la etapa más lenta para operar. Dado que el pipeline entrega un paquete cada S segundos, se tiene un rendimiento máximo (o throughput) = 1/S.

De esta forma, la nueva ecuación aparece como:

$$D = R / S \qquad (1.4)$$

Ejemplo: continuando con el ejemplo anterior, se supondrá ahora que la capa de enlace demora 6 ms, la de red 3 ms, la de transporte 10 ms y la de aplicación 4 ms. El tiempo de respuesta es de 23 ms. El rendimiento es de un trabajo cada 10 ms. El grado de paralelismo es por lo tanto de 23/10 = 2.3 . Es decir, en un momento dado sólo 3 de los cuatro procesadores están activos, los demás están detenidos esperando que sean completadas las etapas de proceso anteriores.

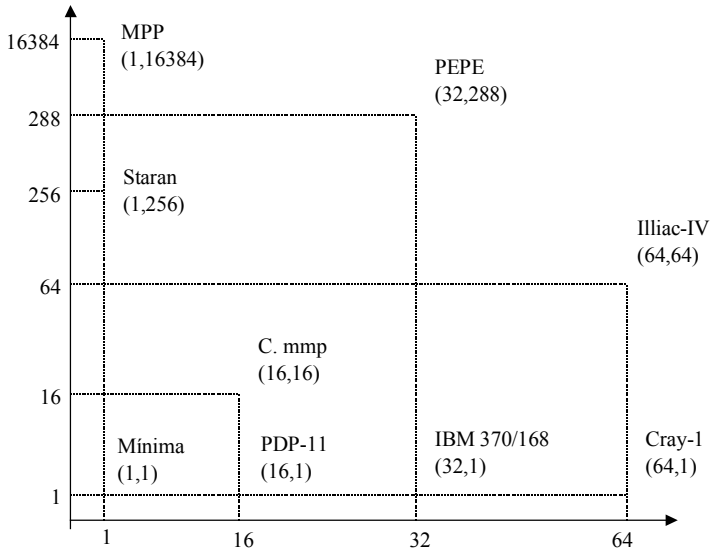

Fig. ***1-9***. Taxonomía de Feng.

Existen cuatro tipos de procesamiento que pueden ser deducidos del diagrama anterior:

- Palabra serie y bit serie (PSBS)
- Palabra paralelo y bit serie (PPBS)
- Palabra serie y bit paralelo (PSBP)
- Palabra paralelo y bit paralelo (PPBP)

El tipo PSBS recibe el nombre de *procesamiento bit-serie* dado que procesa sólo un bit (n = m = 1) cada vez, lo cual resulta en una ejecución bastante lenta. El tipo PPBS (n = 1, m > 1) ha sido llamado procesamiento por sección de bits debido a que procesa un conjunto de bits cada vez. El PSBP (n > 1, m = 1) es el tipo más frecuente de las computadoras actuales, y han sido llamado de *procesamiento por sección de palabra*, dado que procesa una palabra por vez. El esquema PPBP (n > 1, m > 1) es conocido *como procesamiento totalmente paralelo*, en el cual se procesa una matriz de *n* x *m* bits en cada vez; es el modo de ejecución más rápido de los cuatro.

La clasificación propuesta por Wolfgang Händler se basa en identificar el grado de paralelismo y encauzamiento dados por la estructura de hardware de un sistema. No profundizaremos en este modelo dado que se extiende fuera de los alcances de estas notas, pero si haremos mención de que se estudia el procesamiento paralelo encauzado en tres diferentes subsistemas:

- Unidad de control del procesador.
- La unidad Aritmético-Lógica.
- El circuito a nivel de bit.

Sistemas Distribuidos
Ejercicio # 1-2

1. Discuta las ventajas y desventajas relativas de los varios modelos que se usan comúnmente para la configuración de sistemas de cómputo distribuido. ¿Cuál modelo supone que será el más usado en el futuro? Justifique su respuesta.

2. Calcule el grado de paralelismo para una computadora con longitud de palabra de 32 bits y que tiene cuatro cauces aritméticos, cada uno de 16 etapas.

3. Considere el caso de un sistema de cómputo distribuido basado en el modelo de Pool de Procesadores, que tiene P procesadores en el pool. Para este caso, suponga que un usuario comienza un trabajo que consiste en la compilación de un programa compuesto de F archivos fuente ($F < P$). Asuma que en ese momento el usuario es la única persona utilizando el sistema. ¿Cuál puede ser la máxima ganancia en velocidad que puede ser lograda en este caso, comparado con su ejecución en un sistema de un solo procesador (suponga que todos los procesadores tienen igual capacidad)? ¿Qué factores podrían causar que la ganancia en velocidad fuera menor que el máximo?

4. Un servidor de red deber ejecutar cuatro procesos hijo a partir de un proceso padre general. El proceso 1 requiere 3 ms para terminarse, el proceso 2 necesita 15 ms, el proceso 3 requiere 8 ms y finalmente el proceso 4 usa 10 ms. Calcule la razón de incremento si primero ejecutamos en forma serial las tareas con un solo procesador y luego utilizamos una máquina que puede asignar un CPU a cada tarea. ¿Cuál sería la descomposición óptima de la tarea?

5. Estime la tasa de utilización durante 4 segundos de un CPU Pentium de 200 Mhz que tiene un número promedio de bits procesados por ciclo de 18.

6. Suponga que un componente de un sistema distribuido repentinamente falla y analice de que manera afecta a los usuarios del sistema cuando:

a) Se está usando el modelo de Pool de procesadores y el componente que falla es un procesador de dicho pool.
b) Se está usando el modelo de Pool de procesadores y el componente que falla es una terminal de usuario.
c) Se usa el modelo Workstation-Server y falló uno de los servers.
d) Está utilizándose el modelo Workstation-Server y el componente que falló es una de las workstations de usuarios.

Como ya se dijo, otra dimensión de la taxonomía es que en algunos sistemas las máquinas están **fuertemente acopladas** y en otros **débilmente acopladas**. Los sistemas con acoplamiento fuerte, por lo general tienden a utilizarse como sistemas paralelos (es decir, para ejecutar una aplicación en particular) donde hay una sola memoria primaria general (espacio de direccionamiento) que es compartida por todos los procesadores. En tanto, los débilmente acoplados se manejan más como sistemas distribuidos en los cuales los procesadores no comparten memoria y tiene su propio espacio de trabajo. Además de esta distinción, los sistemas débilmente acoplados son capaces de intercambiar datos a la velocidad de su memoria, mientras que los de acoplamiento débil lo hacen a la velocidad de la red de comunicaciones utilizada. La clasificación taxonómica de tales sistemas la aparece en la Fig. 1-10.

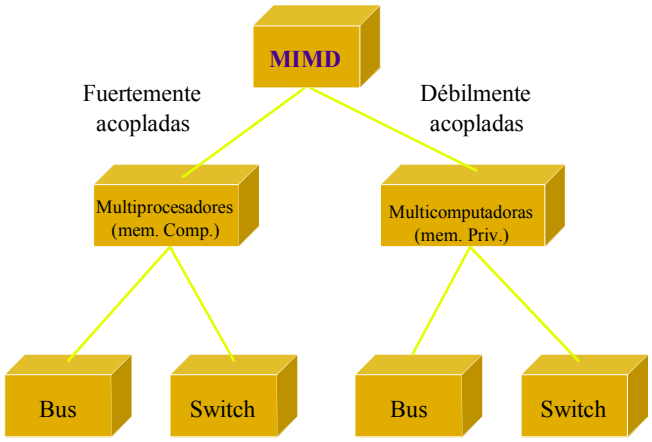

Fig. ***1-10***. Taxonomía de sistemas paralelos y distribuidos.

Se puede resumir lo expuesto diciendo que un sistema de computación distribuida es una colección de procesadores interconectados por una red de transferencia de información en la cual cada procesador posee su propio espacio de memoria y otros periféricos. La comunicación entre dos procesadores se lleva a cabo a través de paso de mensajes sobre dicha red. Para un procesador en particular sus recursos son *locales*, en tanto que otros procesadores y sus recursos son *remotos*. El conjunto de un procesador y sus recursos generalmente recibe el nombre de *nodo*, *site* o *máquina* de un sistema distribuido. Enseguida se explicarán las características de las variantes que presenta esta clasificación.

Multiprocesadores basados en bus.

En este esquema se tienen varios CPU´s conectados a un bus común (típicamente un backplane) junto con un módulo de memoria que es compartido por todos los procesadores en paralelo. Una memoria compartida bajo este esquema se dice que es *coherente*, dado que lo que un procesador escribe en alguna localidad queda inmediatamente disponible para su lectura por otro de los componentes del bus. Para usar la propiedad de localidad en cuanto al diseño, se incrementa la velocidad de acceso en un bus altamente competido añadiendo memoria caché en los CPU´s, de manera que haya una alta probabilidad de localizar un dato previamente leído dentro del caché en lugar de accesar el canal compartido. Con un

tamaño adecuado de caché, se puede incrementar la tasa de reencuentro hacia niveles significativos. Si embargo, el uso de caché trae como consecuencia que la memoria se vuelva *incoherente*, dado que un procesador puede leer información de una localidad de memoria en particular pero a través de su caché, mientras que en la dirección real la información ha sido cambiada por otro procesador.

Una solución propuesta para resolver la problemática es implementar un caché de escritura, sobre el cual se escribe toda palabra que se envíe a memoria. Todos los cachés necesitan realizar un monitoreo constante del bus, de forma tal que cuando cambie el contenido de una dirección de memoria que contienen puedan eliminarlo o actualizar el nuevo valor. Este procedimiento es conocido como *caché monitor*. En la siguiente Fig. 1-11 se muestra un diagrama de tal arquitectura.

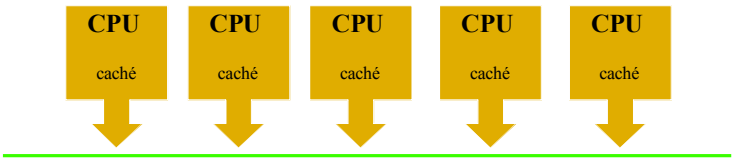

Fig. ***1-11***. Multiprocesadores en bus.

Ejemplo: considerando un sistema multiprocesador construido en base a unidades Pentium de 133 Mhz, que utilizan paquetes de comunicación entre sí con tamaño medio de 500 bytes, y que activar la interrupción para que sea procesado un nuevo paquete consume 10 µs. Asumamos que el código de reenvío de paquetes entre los diferentes procesadores consume 200 ciclos de máquina independientemente del tamaño del paquete, y que se requieren 50 ns para leer o escribir en la memoria que maneja palabras de 4 bytes.

a) ¿Qué tan rápido se pueden reenviar los paquetes si un procesador se involucra en copiar los datos?
b) ¿Cuánto tarda la operación si dicho procesador sólo copia el encabezado de 20 bytes y los datos se transfieren por medio del bus de un CPU al otro?

R: suponer que el ciclo de copiado de información ocurre como sigue

registro ← memoria [read_ptr]
memoria [write_ptr] ← registro
read_ptr ← read_ptr + 4
write_ptr ← write_ptr + 4
counter ← counter – 1
if (counter not 0) salto al inicio del bucle

Suponiendo que las operaciones de escritura y lectura en memoria, así como la de comparación final del contador consumen cada una un ciclo de máquina, y que las tres restantes de actualización de contadores requieren en conjunto sólo un ciclo. De esta forma se tiene que a 133 Mhz, cada ciclo se ejecuta en 7.52 nseg si para leer y escribir en memora

son necesarios además 100 nseg, con lo cal se tiene que cada vuelta al código mostrado se demora: 4 (7.52) + 100 = 130.08 nseg.

a) Un paquete de 500 bytes son 125 palabras de 4 bytes cada una (el tamaño que acepta la memoria). Si se desea copiar todas las palabras serán necesarios:

125 (130.08 ns) = 16.26 μs
el código de reenvío toma 200 (7.52 ns) = 1.504 μs
Tiempo de procesamiento de la interrup. = 10 μs

27.764 μs por cada paquete de 500 bytes.

Lo cual representa más cerca 144.1 Mbps de velocidad de transferencia.

b) Si sólo se copia el encabezado de 20 bytes:

Para palabras de 4 bytes, 20/4 (130.08 ns) = 0.65 μs
el código de reenvío toma 200 (7.52 ns) = 1.504 μs
Tiempo de procesamiento de la interrup. = 10 μs

12.15 μs por cada paquete de 500 bytes.

Lo cual representa más cerca 329.22 Mbps de velocidad de transferencia.

Este ejemplo es un caso totalmente hipotético puesto que ignora ciertos factores reales que se presentan en la práctica como:

- El CPU es un cuello de botella que debe ejecutar otras actividades simultáneamente.
- Se usa una interrupción por cada paquete, cuando en realidad se reciben varios paquetes antes de procesar la interrupción.
- Se supone hay un CPU totalmente dedicado a la comunicación, sin tener la sobrecarga por consulta de tablas de enrutamiento.

Multicomputadoras basadas en bus.

En este sistema no se tiene la compartición de un solo espacio de memoria, sino que cada una de las computadoras conectadas posee su propia memoria local. El sistema de comunicación sirve para interconectar una máquina con otra, con lo que su volumen de tráfico es varios órdenes menor en relación con el uso de una red de interconexión para el tráfico CPU-memoria como ocurre en el caso anterior. Se tratarán mayores detalles de las redes de interconexión en el siguiente capítulo.

Un ejemplo típico de multicomputadoras en bus se encuentra en las modernas redes locales (LAN), en las cuales se agregan tarjetas de red a varios CPU´s que luego se comunican por medio de la infraestructura de cableado. Las velocidades típicas van de 10 a 155 Mbps. Existen casos de sistemas multicomputadoras significativos basados en el esquema.

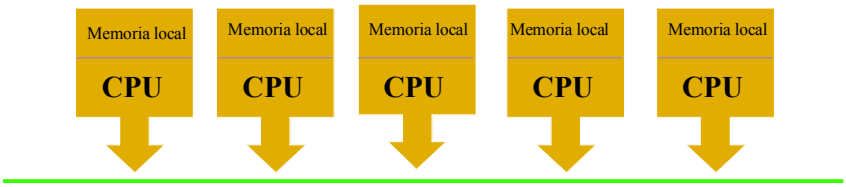

Fig. ***1-12***. Multicomputadoras en bus.

Multiprocesadores basados en switch.

Cuando se llega a un número significativo de procesadores (> 64) se necesita de un mecanismo más eficiente que el bus para lograr conectar cada CPU con la memoria. Una alternativa es dividir la memoria en módulos y luego usar un conmutador matricial como el ilustrado en la Fig. 1-13. De cada CPU y módulo de memoria sale una conexión que puede intersectarse con todas las otras. El hardware cierra pequeños conmutadores que permiten la comunicación de un procesador con un módulo en un momento dado. Una vez terminada la operación de lectura o escritura, dicho switch se libera y quedan las conexiones disponibles para ser usadas en otro nivel de la matriz.

Una ventaja importante es que muchos CPU´s pueden accesar la memoria al mismo tiempo, aunque si dos de ellos desean usar la misma memoria en forma simultánea uno de ellos deberá esperar. Se han hecho modificaciones al cross-bar para minimizar este problema de bloqueo, como el uso de memoria caché en los puntos de cruce o el uso de planificación, pero por el momento no haremos mayores profundizaciones al respecto. La principal problemática del sistema es su tamaño, puesto que con n CPU´s y n memorias, hacen falta n^2 conmutadores en los puntos de cruce. Si n es grande la posibilidad de construcción del equipo se complica. Véase en la Fig. 1-13 un ejemplo de Crossbar de 6 x 6.

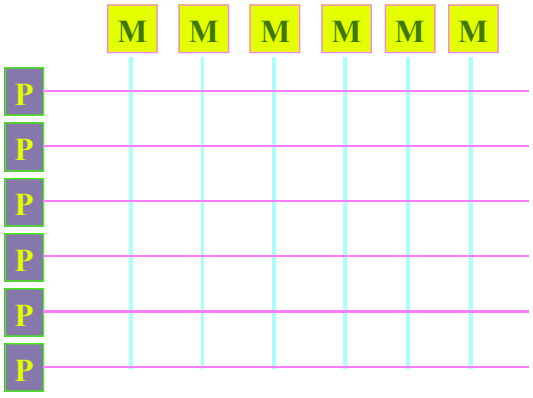

Fig. ***1-13***. Cross-bar 6 x 6.

Originado por los problemas de uso de switches como el Crossbar mencionado, se han desarrollado otras redes de conmutación que requieren menos switches para garantizar una operación confiable del sistema, modificando también la metodología de conmutación seguida. A reserva de profundizar más en el siguiente capítulo, se analizará ahora un caso del sistema de conmutación conocido como Red Omega, en el que se tienen dos niveles de conmutación integrados por switches individuales 2 x 2 (es decir, cada uno tiene dos entradas y dos salidas). A través del diseño del momento de la conmutación, se puede garantizar que un procesador en particular pueda accesar el módulo de memoria que necesita.

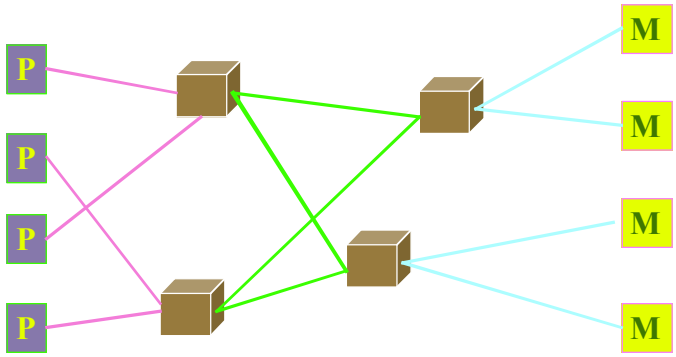

*Fig. **1-14***. Red Omega.

Con ***n*** CPU´s y ***n*** memorias necesita $\mathbf{\log_2 n}$ etapas de conmutación cada una de las cuales tendrá **n/2** switches, para un total de $\mathbf{(n \log_2 n)/2}$ switches.

Ejemplo: calcular el número de etapas de conmutación y la cantidad total de switches que serán necesarios para construir una red omega para 1024 CPU´s.

R:

Etapas de conmutación:	$\mathbf{\log_2 1024} = 10$
Switches por etapa:	$\mathbf{1024/2} = 512$
Switches necesarios:	**5,120**

Si se construyera el equipo con CPU´s de 100 MIPS, cada instrucción debería ejecutarse en 10 nseg. Una petición de memoria, por ejemplo, debe pasar a través de 20 etapas de conmutación (ida y vuelta entre toda la red), por lo que cada conmutador debería operar a la velocidad de 0.5 nseg, o 500 picoseg. El equipo hecho de 5210 switches de 500 picoseg cada uno sería extremadamente caro.

Al igual que en el caso de Crossbar, la red Omega se vuelve cara y difícil de implementar si se presenta un crecimiento elevado del número de procesadores. Se ha intentado reducir el costo mediante sistemas jerárquicos en el que cada procesador tiene asociada cierta memoria, aunque puede seguir accesando la memoria de los otros. Este diseño da lugar a las máquinas NUMA (Non-Uniform Memory Access, acceso no uniforme a la memoria).

Tanenbaum [1] señala el mayor problema de las máquinas NUMA, las que aún con mejor tiempo de acceso presentan problemas para la carga de datos y programas. Además al evaluar la posibilidad de creación de máquinas basadas en multiprocesadores comunicados por Crossbars o por Redes Omega (los cuales siguen siendo caros y lentos) y la dificultad para cargar las máquinas NUMA, el autor concluye que la construcción de grandes sistemas multiprocesadores, fuertemente acoplados y con memoria compartida es difícil y cara.

Con esto se logra otro punto a favor de los sistemas distribuidos, que son principalmente representados por los equipos que discutiremos en la siguiente clasificación.

Multicomputadoras basadas en switch.

Estos sistemas se implementan con base en una sólida red de interconexión a la cual se conectan los CPU´s. Cada equipo tiene acceso directo y exclusivo a su propia memoria. Existen varias estrategias para la conexión topológica, entre las más representativas están la matriz y el hipercubo. Las matrices como la presentada en la Figura. 1-15 se aplican ante todo para problemas de graficación avanzada, procesamiento de imágenes y visión robótica.

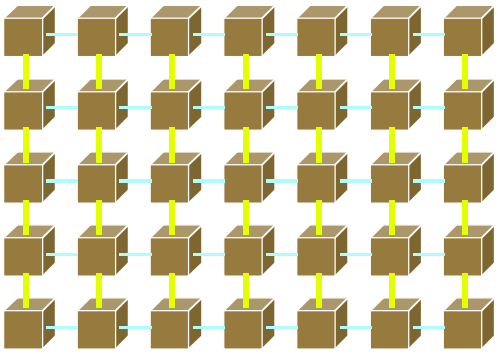

Fig. ***1-15***. Multicomputadora en matriz.

La topología de hipercubo consiste realmente en un cubo *n*-dimensional, la Fig. 1-16 muestra un hipercubo de dimensión 4. Este esquema de conexión puede entenderse como dos cubos ordinarios, con un procesador en cada vértice. Se interconectan los vértices correspondientes de los dos cubos.

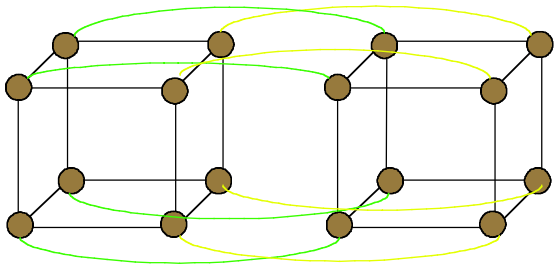

Fig. ***1-16***. Hipercubo de dimensión 4.

Para extender el cubo a 5 dimensiones, se puede añadir a la figura otro conjunto de dos cubos conectados entre sí y conectamos las aristas correspondientes en las dos mitades, y así en lo sucesivo. Para el caso de un hipercubo ***n***-dimensional, cada CPU tiene ***n*** conexiones con otros CPU´s. Como puede apreciarse, la complejidad del cableado aumenta en proporción logarítmica con el tamaño. Dado que sólo se conectan los CPU´s más cercanos, muchos mensajes deben realizar varios saltos antes de llegar a su destino. La trayectoria de mayor longitud igualmente crece en forma logarítmica con el tamaño, a diferencia de la matriz donde ésta crece conforme la raíz cuadrada del número de CPU´s.

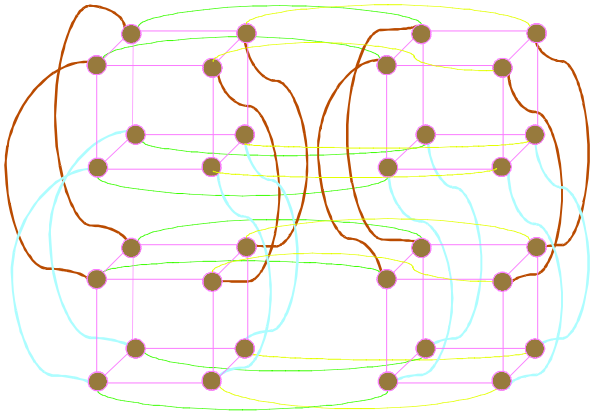

Fig. ***1-17***. Hipercubo de dimensión 5.

Sistemas Distribuidos
Ejercicio # 1-3

1. Lea con su equipo el artículo “Parallel Processing using PVM”, de Richard A. Sevenich y luego “Loki” de Hill, Warren y Goda (*Linux Journal*, No. 45, enero 1998). Discuta con su equipo las arquitecturas (hipercubos, switches, buses) de los sistemas presentados, las diferencias funcionales que exhiben y el tipo de aplicaciones en que son utilizados.

Presenten al resto de la clase los tópicos discutidos, así como sus conclusiones respecto al uso de sistemas distribuidos basados en equipos y software de bajo costo para aplicaciones de alta demanda en nuestros días.

Desarrollen ejemplos de casos de aplicaciones de su trabajo real donde fuera factible la utilización de dichos sistemas.

2. Reelabore el caso de la práctica del multiprocesador Pentium basado en bus que se trató anteriormente, usando las suposiciones y consideraciones que sean necesarias para intentar que el sistema se aproxime a uno de la vida real (tomando en cuenta los detalles señalados).

3. Calcular el número de etapas de conmutación y la cantidad total de switches que serán necesarios para construir una red omega que soportará 256 CPU´s.

4. ¿Con respecto a qué tópicos son mejores los sistemas de cómputo distribuido que los de procesamiento paralelo? Presente y justifique tres aplicaciones donde un ambiente distribuido sea más adecuado para su ejecución que el esquema en paralelo.

5. En un sistema de reservado de boletos para aerolínea se completan 30 tareas en 12 segundos, y cada tarea puede terminarse en 2.3 segundos. ¿Cuál es el grado de paralelismo y qué significa esto?

Tópicos para el diseño de sistemas operativos distribuidos.

En cuanto al software para estos ambientes, la clasificación es mucho más difícil, pero por lo general se asume solamente que existe software débilmente acoplado y fuertemente acoplado. Realmente hay sólo cuatro combinaciones posibles para hardware y software en sistemas distribuidos, dado que el patrón de interconexión es transparente al usuario. Por ejemplo, los sistemas operativos para redes son hardware débilmente acoplado con software débilmente acoplado (puesto que cada estación es autónoma y autocontenida, y sólo hace accesos al sistema cuando requiere algún servicio en particular), en tanto que los sistemas distribuidos son hardware débilmente acoplado sobre software fuertemente acoplado (lo cual logra la imagen de sistema único o "uniprocesador virtual").

Diseñar un sistema operativo distribuido es más complicado que diseñar un ambiente operativo de red por varias razones. En el diseño de un ambiente centralizado se tiene la suposición de que el acceso a los datos del ambiente donde se opera es completo y exacto. Por el contrario, un ambiente operativo distribuido debe ser diseñado con la base de que la información completa del ambiente nunca estará disponible. Se tienen diferentes mecanismos de sincronía y colaboración entre procesos, así como señales de reloj separadas que hacen imposible tener una imagen actualizada del sistema en un instante específico. A pesar de estas dificultades, el sistema operativo debe diseñarse para proveer de todas las ventajas del ambiente de trabajo a los usuarios. Entre las características que deben satisfacerse para el diseño de un sistema distribuido tenemos:

a) Proveer mecanismo de comunicación global entre procesos.
b) Esquema global de protección.
c) Administración de procesos única en todas partes.
d) Sistema de archivos consistente y global.
e) Ejecución de núcleos (kernels) idénticos en todas las computadoras.

Enseguida se discutirán los aspectos claves que hay que cuidar para el diseño de un ambiente operativo, aunque luego se regresará sobre ellos en adelante.

- Transparencia
- Confiabilidad
- Flexibilidad
- Desempeño
- Escalabilidad
- Heterogeneidad
- Seguridad
- Emulación de sistemas operativos existentes

Transparencia

El ambiente operativo distribuido debe volver "invisibles" (transparentes) a todas las máquinas que lo integran y con las cuales trabajan las personas. El requisito es que cada persona trabajando con este conjunto de máquinas conectadas por una red de comunicaciones "sientan" que están trabajando con un único procesador. Existen diferentes enfoques para la transparencia que se presentan enseguida.

Transparencia de acceso.- Por esto se entiende que el usuario no debe distinguir sin un recurso es remoto o local, por lo que respecta al usuario, utilizar un recurso debe ser un mecanismo igual sin importar la localización física de tal ente. Otra forma de expresar esto es diciendo que la interface de usuario –la cual toma forma a través de llamadas al sistema- no debe distinguir entre recursos locales y remotos, y deberá ser responsabilidad del sistema operativo distribuido localizar los recursos y hacer la asignación de los mismos.

Transparencia de localización.- Se entiende básicamente en dos aspectos, los cuales son la *transparencia de nombres* y *movilidad del usuario*. La transparencia de nombre se refiere a que los recursos deben tener nombres únicos dentro del sistema y no estar vinculados con la localización física de los mismos, de manera que se logre que la persona tenga una vista lógica única del sistema. Más aún, los recursos deben conservar un mecanismo y nomenclatura dinámico que se ajusten a los cambios de localización de un nodo a otro dentro del sistema. La movilidad de usuario significa que no importa el punto desde el cual se esté solicitando el acceso a un recurso en particular, la persona siempre tendrá la posibilidad de accesarlo (si las restricciones y condiciones de seguridad lo permiten)

Transparencia de replicación.- A fin de lograr mejor desempeño y confiabilidad, casi todos los sistemas operativos distribuidos crean réplicas de los procesos tanto de los archivos como de los recursos en diferentes nodos del sistema. El nombramiento (nomenclatura) de éstas réplicas, el control de la replicación y de los lugares donde se llevará a cabo deben ser manejados en forma totalmente automática y transparente.

Transparencia a fallas.- El sistema distribuido deberá continuar operando –quizá en forma degradada- aún en el caso de fallas en alguno de sus componentes. Un ejemplo de ello son los sistemas que respaldan recursos como servidores o discos duros, donde todas las acciones se duplican en los componentes, y si el elemento principal falla, los restantes toman su lugar y siguen con la operación sin que los usuarios aprecien la falla. Lo ideal es que esto se logra con la menor sobrecarga (overhead) posible de manera tal que la cooperación entre los diferentes subsistemas no altere el desempeño conjunto esperado. Un sistema totalmente tolerante a fallas aunque teóricamente factible, es muy difícil llevarlo a la práctica: un sistema distribuido totalmente tolerante a fallas sería lento en extremo y demasiado lento dada la enorme cantidad de redundancia requerida para soportar todos los tipos de fallas que pudieran presentarse.

Transparencia de migración.- Significa que los recursos puedan ser cambiados de localización sin alterar su nombre. Hay tres tópicos importantes en este rubro:

a) Las decisiones de migración deben ser hechas automáticamente.
b) La migración de un objeto a otro no requerirá cambio en su nombre.
c) La comunicación interprocesos debe poder enviar un mensaje a un proceso que está siendo reubicado, sin tener que obligar al originador a reenviar de nuevo su paquete de datos.

Transparencia de concurrencia.- Significa que el sistema debe proveer una facilidad de operación tal para compartir sus recursos que cada usuario no perciba la presencia de otros que también están haciendo uso de los mismos.

Transparencia de desempeño.- La capacidad de procesamiento del sistema debe ser uniformemente distribuida entre los procesos existentes en un momento dado. El balance y asignación de cargas de trabajo deben ser automáticamente reconfigurados sin intervención del usuario.

Transparencia de escalamiento.- Esta propiedad permite expandir en escala al sistema sin interrumpir los trabajos de usuarios.

Tipo	Significado
Transparencia de acceso	Poder usar los recursos del sistema independientemente de su localización.
Transparencia de localización	Los recursos tienen nombres consistentes, y pueden usarse desde cualquier sitio que esté el usuario.
Transparencia de replicación	Las réplicas de procesos y recursos se crean y administran en forma automática.
Transparencia a fallas	Las fallas en componentes deben ser resueltas con componentes de respaldo, automáticamente.
Transparencia de migración	Los recursos pueden moverse de lugar y mantener nomenclatura consistente.
Transparencia de concurrencia	Cada persona no nota la existencia de los demás
Transparencia de desempeño	La asignación y balance de carga de trabajo se llevan a cabo sin intervención del usuario
Transparencia de escalamiento	El sistema debe poder expandirse sin interrumpir el trabajo de los usuarios.

*Fig. **1-18***. Transparencia en un ambiente distribuido.

Confiabilidad

En general, se espera que un ambiente distribuido tenga mayor confiabilidad que uno centralizado, dado que se tienen múltiples componentes que funcionan como respaldo. Sin embargo, la existencia de tales componentes por sí misma no puede aumentar la

confiabilidad del sistema. En contraste, el ambiente operativo que administra los recursos debe ser diseñado para incrementar la fiabilidad del sistema tomando partido de las características mencionadas de los ambientes distribuidos. Existen dos tipos generales de fallas que pueden presentarse: en la primera, el sistema se detiene completamente luego de presentarse el estado de error. En la segunda (fallas parciales), el ambiente continúa operando pero arroja resultados erróneos. A continuación describiremos los métodos más comúnmente usados para el tratamiento de fallas.

Eliminación de fallas.- Este criterio se aplica para los aspectos de diseño de componentes del sistema de forma tal que la ocurrencia de fallas sea minimizada. Ciertas prácticas de diseño intentan evadir las fallas mediante el uso de componentes de alta confiabilidad (como equipos certificados, software especial, etc.).

Tolerancia a fallas.- Es la posibilidad de un sistema para continuar operando aún en presencia de problemas en sus componentes. El desempeño del sistema pudiera verse degradado si ciertos elementos críticos se pierden, pero la intención fundamental es que definitivamente no se interrumpa salvo daños extensos. Para la tolerancia a fallas se siguen las técnicas de *redundancia* y *control distribuido*. La redundancia significa evitar los puntos únicos de falla duplicando hardware o software críticos. Esta parece ser a priori una buena idea, puesto que si se tienen dos equipos o programas atendiendo a un servicio en forma simultánea, ante la falla de uno de ellos el segundo continúa atendiendo el servicio específico.

Las técnicas de redundancia imponen una sobrecarga inherente al sistema, para operar las copias del recurso y mantenerlas sincronizadas (coherentes) para cuando sea necesario usar una de ellas. En general, mientras mayor es el número de recursos duplicados, la tolerancia a fallas se minimiza incrementando por ende la confiabilidad, pero añadiendo sobrecarga (overhead) en el sistema. La cuestión que se plantea en la vida práctica es la siguiente: ¿cuánta redundancia es necesaria para cierto problema de incremento en la confiabilidad?

Para resolver el planteamiento, se puede especificar que un sistema es *k-tolerante* a fallas si éste puede continuar en operación aún ante el evento de falla en k componentes. Por tanto, si el sistema debe ser diseñado para tolerar k fallas de interrupción total, necesitamos $k+1$ réplicas. De esta forma, si k réplicas se pierden por fallas, la réplica restante puede ser utilizada para mantener el sistema vivo. Por otra parte, si el sistema debe ser construido para soportar k fallas parciales, serán necesarias un mínimo de $2k+1$ réplicas. Esto debido al sistema de votación que puede usarse para asumir como verdaderos los datos de $k+1$ réplicas cuando k réplicas se comportan anormalmente.

El control distribuido aplica algoritmos de control que pueden ir asignando recursos sobre equipos independientes para aumentar las alternativas en caso de falla de un proveedor de servicio.

Detección y recuperación de fallas.- Aquí se usan métodos basados en hardware y software que pueden determinar la ocurrencia de una falla y luego corregir el sistema hasta un estado estable que permita continuar la operación. Algunas técnicas ampliamente

utilizadas en sistemas operativos distribuidos pueden ser el uso de *transacciones atómicas*, *servers sin estado y transmisión de mensajes basados en reconocimiento y tiempos fuera*.

Una transacción se define como el conjunto de operaciones que se ejecutan en forma indivisible en presencia de fallas y otros procesos en ejecución. Esto es, o todas las operaciones se llevan a cabo de forma exitosa o ninguna de sus operaciones individuales permanece, sin que los otros procesos puedan modificar u observar los estados de dichos cálculos. Si un proceso se detiene en forma inesperada debido a falla en hardware o software, el sistema restaura posteriormente los objetos de datos involucrados hacia sus estados originales.

El server sin estado basa su operación en el enfoque cliente-servidor. Un servidor con estado guarda la información del estado del cliente a lo largo de sus transacciones y afecta la ejecución de la siguiente petición de servicio. Por el contrario, un servidor sin estado no guarda información de sus clientes y puede recobrarse más rápido en caso de alguna falla. Cuando se usa el paradigma de server sin estado, tanto el cliente como el servidor deben tener implementados mecanismos para detectar cuando uno u otro pierden su conexión en el sistema de forma tal que pueden borrar la información de estado y recobrarse de la falla. Aunque realmente la recuperación de fallas se puede dar tanto en server con estado como sin él, es más simple que el sistema se recobre en el segundo esquema y deberá preferirse en la mayoría de los casos.

La transmisión de mensajes basados en reconocimiento y tiempos fuera se basa en que un nodo puede interrumpir la comunicación entre dos procesos en caso de falla del nodo mismo o del enlace de comunicación. En un sistema de comunicación de mensajes, los equipos deben estar en posibilidad de detectar paquetes fuera de secuencia, perdidos o hasta duplicados, así como establecer los *timers* necesarios para no seguir esperando retransmisión de los paquetes faltantes en caso de una falla como las descritas, y hacer el cierre necesario de la conexión.

Tipo	Significado
Eliminación de fallas	Uso de componentes individuales de alta confiabilidad.
Tolerancia a fallas	Los recursos se duplican con técnicas de redundancia
Detección y recuperación de fallas	Determinar la existencia de una falla y luego implementar técnicas para resolverla.

Fig. ***1-19***. Confiabilidad.

Flexibilidad

Esta característica se refleja principalmente en la facilidad de modificación y en la de expansión de las características del ambiente distribuido. La facilidad de modificación intenta resolver los problemas de reemplazo o modificación de componentes debido a errores en el diseño o bien, que el diseño original deba ser cambiado al aparecer nuevos

requisitos por parte de las personas o cambios en el ambiente del sistema. La facilidad de expansión se refiere al mayor o menor grado para añadir nueva funcionalidad al ambiente operativo o a los componentes del sistema en sí.

Tipo	Significado
Flexibilidad de modificación	Cambios o alteración en los componentes.
Flexibilidad de expansión	Añadir nueva funcionalidad.

Fig. ***1-20***. Flexibilidad.

El factor más importante para el diseño del sistema operativo distribuido es el kernel mismo. El kernel es la sección controladora central que provee las facilidades básicas del sistema, opera en una dirección que es inaccesible a los procesos de usuarios, y de hecho constituye la única parte que del ambiente operativo que un usuario no puede reemplazar o modificar. Los dos modelos para el diseño de kernels que se han seguido son el *monolítico* y el microkernel. En el modelo monolítico, muchos de los servicios del sistema operativo como el manejo de la memoria, la planificación, la comunicación entre procesos, el acceso al sistema de archivos y el manejo mismo de los procesos son provistos por el kernel. Esto da como resultado una estructura extensa y compleja.

En el modelo de microkernel la intención es conservar el kernel tan pequeño como sea posible, conformando un núcleo que provea sólo las facilidades mínimas para la instauración de nuevos servicios. Entre las tareas esenciales del microkernel están la comunicación entre procesos, administración de dispositivos en bajo nivel, tareas de planificación y control de la memoria. Todos los demás servicios del ambiente operativo implementan como procesos a nivel del usuario. Un diagrama simplificado de ambos esquemas se presenta en la siguiente Fig. 1-21.

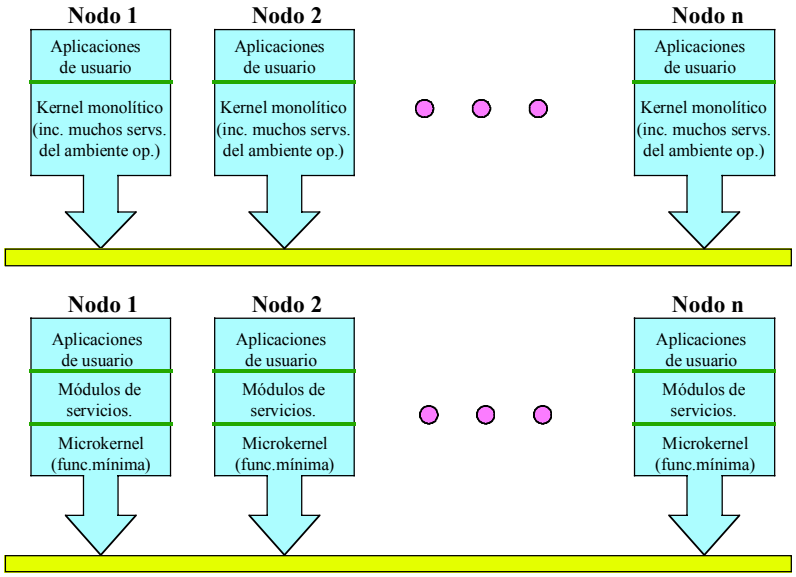

Fig. ***1-21***. Kernel monolítico y microkernel.

El modelo de microkernel tiene las ventajas de: tamaño reducido (lo cual incrementa la flexibilidad y capacidad de configuración), su modularidad por naturaleza y la facilidad de modificar el diseño o añadir nuevos servicios. Más aún en el caso de agregar o cambiar un servicio no hay necesidad de detener el sistema y construir un nuevo kernel, como sucede en los sistemas monolíticos. Por el otro lado, entre sus desventajas están considerar que el paso de mensajes a través del kernel para toda la comunicación entre procesos impone una sobrecarga natural a diferencia del modelo monolítico (donde el mismo espacio es compartido por todos los servicios y el kernel) donde no se requiere paso de mensajes ni conmutación de contexto.

Desempeño

La consideración en este punto es que el desempeño del ambiente distribuido debe ser al menos tan bueno como el centralizado, para lo cual se requiere que los componentes sean construidos de forma específica. Sus principios fundamentales son:

Usar batching donde sea posible.- El uso de proceso en batch siempre mejora el rendimiento en el ambiente distribuido. Por ejemplo, es mejor transmitir grandes segmentos de datos a través de la red que enviar páginas individuales. El uso de *piggybacking* (asentimientos de mensajes anteriores dentro del siguiente mensaje de una serie) también apoya el incremento del desempeño.

Usar batching tiene las siguientes ventajas importantes con respecto al tiempo de ejecución de procesos:

- La sobrecarga de tareas se incrementa en factor menos que lineal con el número de tareas que se van agrupando.
- El tiempo necesario para acumular un lote no es extenso.

Para presentar con mayor detalle estas ideas se definen las variables S → unidades de tiempo para ejecutar una tarea, y O → sobrecarga por ejecución en unidades de tiempo. Suponiendo que el sistema procesa una tarea inmediatamente en cuanto llega, si en ese instante el sistema está desocupado se tendrá:

$$\text{Caso de por tiempo de respuesta para la tarea} = S + O \qquad (1.5)$$

$$\text{Peor caso de máximo rendimiento} = \frac{1}{S+O} \qquad (1.6)$$

Asumiendo ahora que se está formando un lote con N tareas que requieren un tiempo A para llegar al sistema, y que la sobrecarga para las N tareas es P, de manera tal que $P < N*O$. Si las tareas en el lote se atenderán en base a la rutina FIFO, se tendrá:

$$\text{Caso de por tiempo de respuesta para la tarea} = A + (N*S) + P \qquad (1.7)$$

$$\text{Peor caso de máximo rendimiento} = \frac{N}{(N * S) + A + P} \tag{1.8}$$

La ecuación (1.8) puede escribirse finalmente como

$$\text{Peor caso de máximo rendimiento} = \frac{1}{S + \left(\frac{A+P}{N}\right)} \tag{1.9}$$

Con lo que el incremento en máximo en rendimiento cuando se usa batching es:

$$\frac{S+O}{S + \left(\frac{A+P}{N}\right)} \tag{1.10}$$

Esta relación es mayor que 1 cuando $A < (N*O - P)$, esto es, el tiempo consumido en acumular N paquetes es menor que el tiempo ganado en la sobrecarga decrementada.

Durante el tiempo A el sistema es libre de ejecutar otras tareas, de forma tal que si fuese medido el rendimiento solamente durante el tiempo que el equipo se dedica a dicha tarea se obtendría

$$\text{Peor caso de máximo rendimiento} = \frac{1}{S + P/N} \tag{1.11}$$

el cual es siempre mayor que el rendimiento sin batching.

Ejemplo: al recibir un paquete una tarjeta de red generalmente enciende una interrupción para avisar al CPU que debe realizar una tarea. El tiempo transcurrido en asignar la interrupción puede significar una sobrecarga significativa en el procesamiento del paquete, lo cual puede motivar a usar técnicas de "batching". Una estrategia común en tanto se agrupan interrupciones es encender una interrupción con un retardo fijo luego de recibir un paquete. Esto limita el peor tiempo de respuesta, mientras incrementa el rendimiento si llegan varios paquetes uno tras otro.

Suponiendo que la tarjeta interrumpe al CPU 10 ms luego de la llegada de un paquete, haya o no otro paquete en el lote. Si se establece que el tiempo de procesamiento por paquete es de S = 2 ms, y que la sobrecarga de asignación de interrupciones es O = 5 ms. Entonces el peor tiempo de respuesta sin lotificación es de 5 + 2 = 7 ms, y con ella sería de 10 + 2 + 7 = 17 ms. Sin lotificación el mejor rendimiento alcanzable es de 1 paquete cada 7 ms = 0.143 paquetes/seg . Sin embargo si la tasa promedio de llegada de paquetes es $\lambda = 1 paq / seg$, se esperaría entonces tener $\lambda A = 10$ paquetes en un lote, mejorando el rendimiento en 10 paquetes cada (10 + 2 *10 +5) = 35 ms = 0.285 paquetes/ms.

Ignorando el tiempo de espera de los paquetes, este resultado se mejora en 10 paquetes cada 25 ms = 0.4 paquetes/ms.

Ejemplo: una sesión de terminal remota (TELNET) envía un paquete por cada carácter tecleado, lo cual parece ser ineficiente. Ud. debe diseñar una aplicación que aplique batching en los caracteres. Asumiendo que los usuarios teclean caracteres a la tasa pico de 5 caracteres/seg, y con un promedio de 1 carácter/seg.

a) ¿Si se pueden retardar los caracteres no más de 500 ms antes de que deban enviarse cuál es el tamaño del lote más largo y el tamaño promedio?

b) Si el tamaño del lote se fija a 20 caracteres, cuáles son los retardos promedio y mejor que se pueden lograr a nivel de aplicación?

c) ¿Qué le dice todo esto acerca de usar batching para TELNET?

Soluciones:

a) El tamaño más grande del lote es cuando los paquetes llegan en el valor pico de 5 caracteres/seg = 200 ms/carácter. Por tanto, el tamaño más grande es 500/200 = 3 caracteres. El tamaño promedio del lote es 500/1000 = 1 carácter.

b) Si el tamaño del batch es de 20 caracteres, el mejor tiempo de respuesta es de 20 caracteres/5 caracteres/seg = 4 segundos. El tiempo de respuesta promedio es de 20/1 = 20 segundos.

c) Si las suposiciones hechas son correctas, no conviene hacer un TELNET con lotes.

Tópicos de diseño:

Usar caché donde sea posible.- El uso de caché vuelve a los datos más disponibles, además de que puede reducir la contienda por recursos centralizados.

Minimizar la copia de datos.- El copiado de datos involucra operaciones costosas en términos de tiempo en CPU, de manera que sería deseable minimizar hasta lo posible dicha operación, aunque en la realidad no siempre puede lograrse tal fin. Por medio de una administración adecuada de la memoria se puede eliminar mucho del movimiento de datos entre el kernel, los bloques de E/S, clientes y servers.

Minimizar el tráfico en la red.- Como el sistema distribuido está sustentado en la red de comunicaciones, mantenerla libre de tráfico se refleja en el rendimiento integral del sistema. Minimizando el tráfico inter-nodos de procesos que deban operar de forma colaborativa (quizá acercándolos físicamente sobre la red) o reuniéndolos en un solo nodo se puede minimizar dicho factor.

Tipo	Significado
Usar batch donde sea posible	El batching siempre mejora el rendimiento en un sistema distribuido.
Usar caché donde sea posible	Una forma de tener datos más disponibles y reducir contienda por recursos centralizados
Minimizar la copia de datos	A fin de no tener tiempo gastado de CPU
Minimizar el tráfico en la red	

Fig. ***1-22***. Flexibilidad.

Escalabilidad

Se refiere a la capacidad del sistema para adaptarse a una demanda incrementada de servicio. Lo más común es que los ambientes distribuidos evolucionen y crezcan tanto en usuarios como en servicios, por lo tanto es deseable que pueda adaptarse a estas nuevas especificaciones sin causar interrupción de los servicios o pérdidas de desempeño significativas. Algunas reglas de diseño para la escalabilidad las mencionaremos enseguida.

Evitar entidades centralizadas.- El uso de una entidad centralizada como una base de datos o un servidor de archivos en especial vuelve al sistema no escalable debido a las siguientes razones:

a) La falla en el componente central causa la caída total del servicio en el sistema.
b) El desempeño de un componente centralizado se convierte en un cuello de botella cuando se tienen más usuarios compitiendo por él.
c) Aunque el componente centralizado tenga la suficiente capacidad de desempeño, almacenamiento o velocidad, la red puede saturarse en las regiones cercanas a dicha entidad, cuando el tráfico se incrementa por el acceso a dicho recurso.
d) En una red amplia, formada por varias redes interconectadas, es poco eficiente tener un tipo particular de peticiones atendido sólo en un punto.

En general deberá evitarse el uso de componentes centralizados, usando replicación de recursos y algoritmos de control para satisfacer éste objetivo de diseño.

Evitar algoritmos centralizados.- Un algoritmo centralizado puede definirse como aquél que recolecta información de varios nodos, la procesa en uno solo y luego distribuye los resultados entre los demás. Su uso debería ser restringido por razones muy similares a las del punto anterior, es decir la implementación no opera suficientemente bien con el incremento de peticiones de servicio. En lugar de ello, un algoritmo descentralizado no recolecta información global del sistema, y las decisiones en los nodos se toman sólo sobre la base de información global.

Ejecutar la mayoría de las operaciones en las estaciones clientes.- Esta técnica favorece la escalabilidad, puesto que realiza una degradación suave del desempeño del sistema conforme crece en tamaño, a través de la reducción de la contienda por recursos compartidos.

Tipo	Significado
Evitar entidades centralizadas	El batching siempre mejora el rendimiento en un sistema distribuido.
Evitar algoritmos centralizados	Una forma de tener datos más disponibles y reducir contienda por recursos centralizados
Ejecutar la mayoría de las operaciones en las estaciones clientes	A fin de no tener tiempo gastado de CPU

Fig. ***1-23***. Escalabilidad en sistema operativo distribuido.

Heterogeneidad

La heterogeneidad es un factor importante para la construcción de un ambiente distribuido, dado que en este tipo de sistemas lo más probable es que se tengan conectados diferentes tipo de plataformas de cómputo las cuales deben tener resueltas las incompatibilidades que presentan entre sí (como diferencias de formatos de archivos, longitud de palabra de los equipos y representación interna de caracteres). Para implementar la solución de esta problemática generalmente lo que se hace es construir software de traducción que permite convertir la información de un ambiente hacia otro. En lugar de implementar todas las alternativas de conversión para los equipos que se tienen instalados, lo recomendable es generar un formato de intercambio estándar para todos, de forma tal que cada plataforma sólo necesite el desarrollo del sistema de traducción de sus datos hacia dicho formato estándar y viceversa.

Seguridad

Para que las personas puedan confiar en la integridad del sistema y la operación que con él puede hacerse, debe implementarse toda la protección contra accesos no autorizados y destrucción total o parcial tanto de hardware como de software. A diferencia de los sistemas centralizados en donde la autentificación de un usuario o programa que intenta hacer uso de un recurso en particular se ejecuta en la misma máquina a la que se solicita el acceso, en un sistema distribuido dicha información no puede garantizarse en cuanto a veracidad.

En particular, se tienen los siguientes requerimientos:

a) Debe ser posible para el originador de un mensaje saber que dicha información llegó al receptor deseado.

b) Debe ser posible para el receptor de un mensaje determinar que éste fue emitido por el originador genuino.
c) Tanto el receptor como el transmisor deben implementar mecanismos para asegurarse que la información original no ha sido alterada durante su tránsito de un extremo a otro

Emulación de sistemas operativos existentes

Un sistema operativo distribuido debería implementar emulación de sistemas operativos populares –como por ejemplo UNIX-, de manera que el nuevo software desarrollado pudiera hacer llamadas a la interface del ambiente distribuido y tomara partido de sus funciones.

Ejemplos de Sistemas Distribuidos.

Hasta ahora se ha mencionado que existen diferentes tipos de sistemas distribuidos, en este apartado se mencionan algunos de ellos y las características que los distinguen, téngase presente que el texto se refiere a los siguientes tipos de sistemas distribuidos, de acuerdo a la clasificación de Langsford [2]:

- Kernels de sistemas distribuidos con mínima funcionalidad.
- Sistemas integrados.
- Sistemas orientados a objetos
- Sistemas basados en el modelo de pool de servers.

Kernels de sistemas distribuidos con mínima funcionalidad.

Los sistemas en tiempo real pueden estar tanto basados en el modelo de pool de servers como en el de orientación a objetos. Los kernels de funcionalidad mínima –como ya se dijo- reciben el nombre de microkernels, dado que proveen un conjunto casi mínimo de primitivas para la administración de procesos, manejo de memoria, intercambio de mensajes y la administración de la memoria de respaldo. Ejemplos de éstos sistemas incluyen: ACCENT y MACH de la universidad Carnegie Mellon, V KERNEL de Stanford y QNX, de QNX Software Ltd.

Sistemas integrados.

Estos tipos de sistemas permiten la ejecución de software estándar ampliamente configurable en cualquier computadora, y provee a cada máquina de las ayudas y servicios que necesita para tal función. Actualmente están basados fundamentalmente en el sistema operativo UNIX y corren sobre equipos que lo ejecutan, pero existen también diversos sistemas previos a las funciones de UNIX como LOCUS que fue desarrollado por la universidad de California en Los Angeles y ahora se vende por Locus Inc., D-UNIX de los laboratorios Bell, SAGUARO, de la universidad de Arizona y Galaxy, de la universidad de Tokio.

Sistemas orientados a objetos.

Tratan a las aplicaciones como objetos abstractos sobre los cuales ejecutan operaciones abstractas. Ejemplos: ARGUS del MIT, EDEN de la universidad de Washington y AMOEBA de la Vrije Universiteit Amsterdam.

Sistemas basados en el modelo de pool de servers.

Utilizan un server con los servicios disponibles para cada usuario. El Sistema Operativo Distribuido Cambridge es un caso de esta clase.

Introducción a DCE (Distributed Computing Environment).

DCE es un esfuerzo de la Open Software Foundation (OSF) por crear un ambiente de cómputo distribuido independiente de los vendedores. A la OSF pertenecen empresas como IBM, DEC, HP, etc., quienes han aportado diferentes conceptos y mecanismos para integrar DCE, el cual no puede considerarse ni como ambiente operativo ni como aplicación.

DCE es realmente un conjunto integrado de servicios y herramientas que puede ser instalado como un ambiente coherente sobre sistemas operativos existentes y servir como plataforma para la construcción y ejecución de aplicaciones distribuidas. La fundamentación de que este ambiente sea independiente de los vendedores es que pueda correr sobre diferentes equipos, sistemas operativos y redes. Por ejemplo, algunos sistemas operativos hacia los cuales se puede transportar DCE incluyen OSF/1, AIX, DOMAIN OS, ULTRIX, HP-UX, SUN OS, UNIX SYSTEM V, WINDOWS y OS/2.

Como se ve en la Fig. 1-24, DCE es *middleware* estructurado por niveles entre la capa de aplicaciones DCE y la capa del sistema operativo y la red. La idea básica es tomar una colección de máquinas existentes (posiblemente de diferentes fabricantes), interconectarlas en red y agregar la plataforma del software DCE sobre cada sistema operativo, lo cual permitirá construir y ejecutar las aplicaciones distribuidas. Cada máquina conserva su propio ambiente operativo local, y a través de la capa de software DCE elimina sus diferencias con las otras (ejecutando las conversiones de tipos de datos que sean necesarias) de forma tal que se logra hacer transparente para los programadores a un ambiente básicamente heterogéneo.

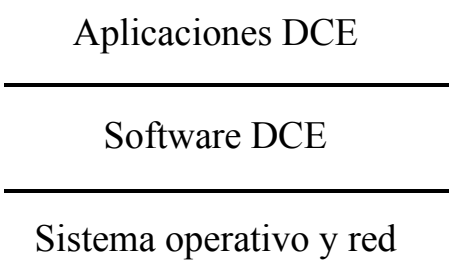

Fig. ***1-24***. Esquema de DCE.

DCE no fue creado por OSF, sino que pasó por un proceso de selección entre muchas otras propuestas cuando la OSF abrió la licitación a universidades e industria para buscar las herramientas y servicios necesarios para la construcción de un ambiente distribuido de cómputo que fuera coherente. El código de servicios y herramientas que se aceptó recibió más modificaciones por parte de OSF hasta que se logró un solo paquete integrado que estuvo disponible en enero de 1992 como DCE v 1.0.

Componentes de DCE

Paquete de hilos.- Este paquete provee un modelo de programación sencilla para la construcción de aplicaciones concurrentes. Incluye operaciones para crear y controlar varios hilos de ejecución en un solo proceso y para el acceso a datos globales desde las aplicaciones.

Facilidad para llamadas a procedimientos remotos (RPC).- Es la base para comunicaciones en DCE, su simplicidad de uso e independencia de los protocolos de red proveen una plataforma para comunicaciones seguras entre cliente y servidor. Por otra parte, oculta las diferencias entre equipos mediante la conversión de datos hacia los formatos necesarios por el cliente y el server.

Servicio de tiempo distribuido (DTS).- Se utiliza para sincronizar la referencia de tiempo de todas las computadoras que pertenecen al sistema distribuido, ya sea tomando como base el reloj de alguna de ellas o alguna fuente externa, inclusive de algún otro sistema similar.

Servicio de nombres.- Este permite nombrar en forma unívoca y transparente a recursos como servers, archivos, dispositivos, de manera totalmente independiente de su localización física. Este servicio incluye el Cell Directory Service (CDS), el Global Directory Service (GDS) y el Global Directory Agent (GDA) como elementos para brindar esta funcionalidad.

Servicio de seguridad.- Provee todas las herramientas para autentificación y autorización de accesos para proteger los recursos contra uso no autorizado.

Servicio de archivos distribuidos (DFS).- Es un sistema de archivos a nivel general del ambiente, que posee características como transparencia de localización, elevado rendimiento y alta disponibilidad. Su principal característica es que también puede otorgar acceso a clientes de otros sistemas de archivos.

Otra finalidad importante es que DCE sea escalable, de tal forma que soporte miles de usuarios y equipos operando sobre redes de cobertura amplia. Para lograr este nivel de funcionalidad se usa la operación por celdas (un conjunto de personas, máquinas y recursos que tienen propósitos comunes y usan recursos compartidos). Una celda mínima necesita un server de directorio para ella, un server de seguridad, un servidor de tiempo distribuido y una o más estaciones clientes con procesos correspondientes para cada tipo de servidor. Se tiene toda la libertad para definir una celda, de acuerdo a especificaciones de trabajos a realizar en ella y ciertas tareas administrativas, y para ello se siguen cuatro factores que apoyan a definición en particular de esta unidad operativa:

Propósito: los equipos de las personas que cumplen funciones comunes o que accesan recursos similares pueden constituir un tipo de celda.

Administración: los equipos sobre los cuales se aplican políticas similares en cuanto a control de recursos pueden simplificar el manejo del sistema y servir como criterio de integración.

Seguridad: las computadoras de usuarios que dependen del acceso a otros para su proceso pueden determinar una celda, donde los límites de la misma actúan como un *firewall* mantener los recursos de la celda libres de intromisión, de forma que para usarlos deberá pasarse por un mecanismo de autentificación y autorización específico.

Sobrecarga: ciertas operaciones DCE como resolución de nombres y autentificación de usuarios imponen una sobrecarga elevada al sistema. Así que colocando en la celda a los equipos de los cuales se depende mutuamente para estos procesos y otros que operan en forma similar se logra reducir el impacto global dejando el flujo de la transacción operando sólo a nivel local.

Sumario

En esta primera parte se han presentado los conceptos esenciales de los sistemas distribuidos, sus características básica y las causas tecnológicas/funcionales que determinaron su aparición y desarrollo. Se analizaron también los diferentes modelos para la creación de cómputo distribuido y los tópicos más importantes que rigen el diseño de un sistema operativo distribuido, que es el software fundamental que mantiene en operación a estos sistemas.

Por último, fue presentada una breve introducción a DCE, un ambiente de cómputo distribuido abierto, que intenta ser la plataforma común para el diseño de aplicaciones distribuidas sobre plataformas heterogéneas de cómputo.

Sistemas Distribuidos
Ejercicio # 4

1. Se ha mostrado que se usa mucho el enfoque de organización por capas funcionales en lo que respecta al diseño de sistemas operativos distribuidos. Elabore una discusión sobre las principales ventajas de utilizar dicho esquema.

2. Analice con su equipo de trabajo los más importantes tópicos que puede seguir una diseñadora o diseñador de un sistema operativo distribuido para incrementar la confiabilidad de su sistema. ¿Cuál es el principal problema que se presente para construir un sistema confiable?

3. ¿Cuáles son los principales problemas que aparecen en sistemas distribuidos heterogéneos? ¿Cuáles son las principales problemáticas que tienen que resolverse para satisfacer la heterogeneidad?

4. Mencione las principales cuestiones que deben cuidarse en el diseño de un sistema operativo distribuido para garantizar un adecuado nivel de desempeño.

5. Instale en su computadora el sistema operativo QNX basado en microkernel (el instructor le proveerá con el software necesario). Desplácese por la interface gráfica del sistema y lea las notas de desarrollo tanto del sistema operativo en sí como del servidor web (httpd) y navegador web que se incluyen. Tome nota de las características del ambiente y de su desempeño, así como de la interface, aplicaciones y ayudas que provee al usuario.

6. Luego de terminar el punto anterior, discuta con su equipo de trabajo la conveniencia de los microkernels. ¿Cuáles son las principales tareas que debe desempeñar un microkernel? Elaboren un informe de ventajas relativas entre el enfoque de microkernel y kernel monolítico.

Sistemas Distribuidos
Tarea # 1

Nombre:

_____

1. Leer el artículo "The scientific workstation of the future may be a pile of PC´s", de Thomas Sterling (*Communications of the ACM*, september 1996, Vol. 39. No. 9) . Elaborar un comentario crítico al respecto.

2. Lea el artículo "Parallel Computing using Linux", de Manu Konchady (*Linux Journal*, No. 45, enero 1998).
Elabore una contrastación entre los conceptos presentados en dicho texto y aquellos que hemos analizado hasta el momento.

3. Leer el artículo de Daryll Strauss, "Linux helps bring Titanic to life". Elaborar un comentario crítico al respecto. Discuta los típicos de la arquitectura usada (dónde se clasificaría ésta) y el tipo de sistema distribuido que se usó en el proyecto.

4. Una multicomputadora con 256 CPU´s se organiza como una matriz de 16 x 16. ¿Cuál puede ser el mayor tiempo de retraso (correspondiente a los saltos) para un mensaje?

5. Considerando el caso de un hipercubo de 256 CPU´s. ¿Cuál sería ahora el máximo tiempo de retraso correspondiente a los saltos? ¿De qué nivel sería ese hipercubo?

6. Si tenemos un multiprocesador de 4096 CPU´s de 50 MIPS conectados a la memoria por medio de una red omega, ¿con qué rapidez deben permitir los switches que una solicitud vaya a la memoria y regrese en un tiempo de instrucción?

7. Compare los siguientes tipos de sistemas en términos de su costo, complejidad de hardware, complejidad del ambiente operativo, paralelismo potencial y *programabilidad* (qué tan fácil es escribir programas eficientes para ellos).

a) Un sistema multiprocesador con una sola memoria compartida.
b) Un sistema multiprocesador en el cual cada procesador tiene su propia memoria local, además de la memoria compartida con todos los demás procesadores.
c) Un sistema de multiprocesador en el cual todos los equipos están en un gran cuarto, e interconectados por una red de comunicación de alta velocidad conformando una red. Cada procesador se puede comunicar con los otros únicamente mediante el paso de mensajes.
d) Un sistema multiprocesador donde cada procesador tiene su propia memoria. Los procesadores están situados muy lejos unos de otros (posiblemente en diferentes ciudades) y están interconectados por un sistema de baja velocidad conformando una red. Cada procesador se puede comunicar con los otros únicamente mediante el paso de mensajes.

Como guía para la comparación considere los siguientes casos: (a) el número de procesadores es pequeño (2-8); (b) el número de procesadores es grande (16-32); y (c) el número de procesadores es muy grande (> 100).

Fecha de entrega:
Entregar esta hoja como carátula del trabajo.