# INDICE GENERAL

## Contenido

## ÍNDICE DE FIGURAS

# 1 - INTRODUCCION

Los sistemas distribuidos están formados por un conjunto de computadoras interconectadas. Aunque cada una de estas máquinas se encuentra en ubicaciones físicas distintas, y tiene su propio hardware y software, todas comparten una red de comunicación común que las enlaza. De esta manera, el programador puede verlas como un único sistema, pero con múltiples componentes, experimentando una distribución transparente que opera de manera coordinada y sincronizada.

Desde una perspectiva informática, un sistema distribuido es aquel cuyos elementos, ya sean de hardware o software, están alojados en diferentes nodos de una red. Estos elementos se comunican y se sincronizan mediante el intercambio de mensajes.

El objetivo principal de este trabajo es exponer los aspectos clave que forman parte de los sistemas distribuidos, analizando algunos de los conceptos esenciales del software distribuido, como la arquitectura cliente/servidor, la transmisión de mensajes y las llamadas a procedimientos remotos. También se revisa la importancia de la arquitectura en clústeres.

# 2 – DESARROLLO

## 2.1 – COMPUTACION CLIENTE/SERVIDOR

### 2.1.1 – ¿Qué es la Computación Cliente Servidor?

Como el propio término sugiere, un entorno **"Cliente / Servidor"** está formado por clientes y servidores:

- Las máquinas cliente son normalmente simples “PCs” o **"Estaciones de Trabajo"** que proporcionan una interfaz de fácil manejo al usuario final.
- Los servidores son un entorno **"Cliente / Servidor"** que brinda un conjunto de servicios compartidos a los clientes.

La computación **"Cliente / Servidor"** normalmente es una computación distribuida, cuyos usuarios, aplicaciones y recursos se distribuyen en respuesta a los requisitos de trabajo, y se unen a través de una **"LAN"**, **"WLAN"** o **"Internet"**.

Algunos términos característicos de productos **"Cliente / Servidor"** son:

- **Interfaz de programación de aplicaciones (API)**: Un conjunto de funciones y programas que permiten a los clientes y servidores intercomunicarse.
- **Cliente**: Un elemento de la red que solicita información, normalmente un “PC” o **"Estación de trabajo"**. Puede interrogar a una base de datos o solicitar información de un servidor.
- **Middleware**: Un conjunto de controladores, “API”, y software adicional que mejoran la conectividad entre una aplicación cliente y un servidor.
- **Base de datos relacional**: Una base de datos en la que el acceso a la información está restringido por la selección de filas que satisfacen todos los criterios de búsqueda.
- **Servidor**: Un computador, normalmente una estación de trabajo de gran potencia, un minicomputador, o un mainframe, que almacena la información para los clientes de la red.
- **Lenguaje estructurado de Consultas (SQL)**: Lenguaje desarrollado por IBM y estandarizado por ANSI que permite acceder, crear, actualizar e interrogar bases de datos relacionales.


Figura 01. Esquema del Modelo Cliente Servidor.

### 2.1.2 - ¿QUÉ LO DIFERENCIA DE OTRAS SOLUCIONES DE PROCESAMIENTO DISTRIBUIDO?

- Es fundamental que los usuarios cuenten con aplicaciones intuitivas en sus equipos, lo que les otorga mayor control sobre el uso de sus recursos y permite a los administradores responder eficientemente a las demandas locales.
- La clave está en centralizar tanto las bases de datos corporativas como las herramientas de gestión de red, aunque las aplicaciones se encuentren distribuidas. Este enfoque facilita a los administradores un control más efectivo y mejora la interoperabilidad entre sistemas, aliviando a los departamentos de TI de gran parte del trabajo de mantenimiento.
- Se apuesta por mantener un sistema abierto y modular, ofreciendo a los usuarios la flexibilidad de elegir y combinar productos de diferentes proveedores según sus necesidades.
- La administración y la seguridad de la red son prioritarias en la estructura y operación de los sistemas de información, garantizando un entorno fiable y bien gestionado.

### 2.1.3 – APLICACIONES CLIENTE/SERVIDOR

La característica fundamental de una arquitectura “Cliente / Servidor” es la distribución de las tareas de la aplicación entre el cliente y el servidor.

La plataforma y el sistema operativo del cliente y del servidor pueden ser diferentes. De hecho, pueden existir diferentes plataformas de clientes y sistemas operativos y protocolo de comunicación y soportar las mismas aplicaciones.

Quien permite que interactúen el cliente y el servidor es el software de comunicaciones. El principal ejemplo de este software es **"TCP/IP".**

El diseño de la interfaz de usuario es sumamente decisivo en la máquina cliente, para que esta sea un éxito en el entorno **"Cliente / Servidor",** ya que es como el usuario interactúa con el sistema.

### 2.1.4 – APLICACIONES DE BASES DE DATOS

Para explicar el concepto utilizaremos las bases de datos relacionales como ejemplo de una de las aplicaciones de tipo cliente/servidor. En este entorno, el servidor es básicamente un servidor de base de datos. Mediante transacciones es que interactúan el cliente con el servidor, en las que el cliente hace una apetición a la base de datos y recibe una respuesta.

Como el servidor es el responsable del mantenimiento de la base de datos se requiere un complejo sistema gestor de base de datos y lo que une al cliente y al servidor es el software que hace posible que el cliente realice peticiones para acceder al servidor de la base de datos, como lo es, por ejemplo, el lenguaje estructurado de consultas (SQL) **[1].**

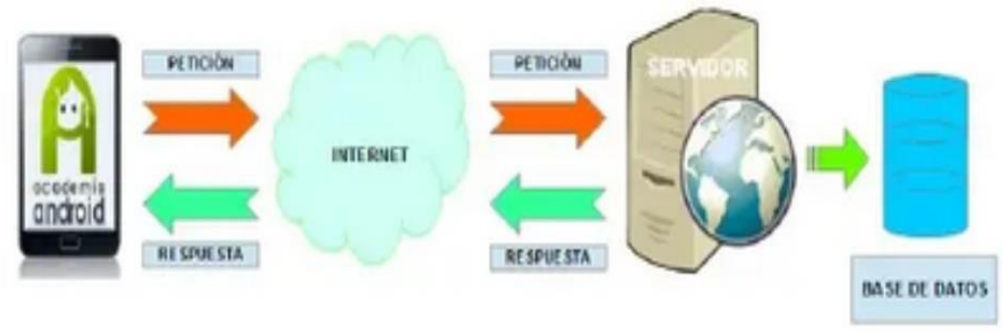


Figura 02. Arquitectura cliente/servidor para aplicaciones de bases de datos.

### 2.1.5 – CLASES DE APLICACIONES CLIENTE/SERVIDOR

Hay una serie de implementaciones que dividen el trabajo entre el cliente y el servidor de diferentes maneras que podríamos analizar a continuación:

Procesamiento basado en el HOST: Se refiere a los entornos “mainframe” tradicionales en los que virtualmente todo el procesamiento se realiza en el host central. No es una verdadera computación cliente/servidor como tal. menudo, la interfaz de usuario se realiza a través de un interfaz tonto. Incluso si el usuario está empleando una computadora, la estación del usuario se limita al papel de emulador de terminal **[2].**

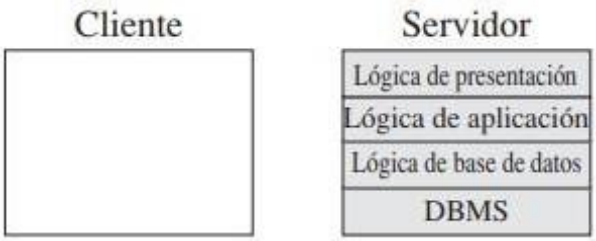


Figura 03. Procesamiento basado en el host.

- Procesamiento basado en el SERVIDOR: En esta configuración el cliente es el responsable de proporcionar la interfaz gráfica de usuario, mientras todo el procesamiento se realiza en el servidor. Es una configuración típica en sistemas a nivel de departamento. Sin embargo, este tipo de configuraciones no suele generar grandes ventajas en productividad, ni cambios fundamentales en las funciones de negocios soportadas por el sistema.

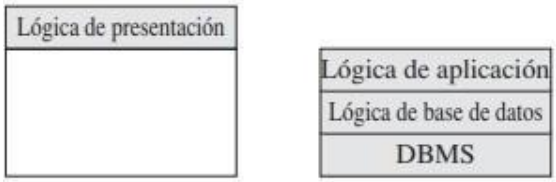


Figura 04. Procesamiento basado en el servidor.

- Procesamiento basado en el CLIENTE: Contrariamente al caso anterior, todo el procesamiento se puede realizar en el cliente, con excepción de las rutinas de validación de datos y otras funciones de la lógica de la base de datos que se pueden realizar mejor en el servidor. Esta arquitectura permite a los usuarios el uso de las aplicaciones adaptadas a las necesidades locales **[2].**

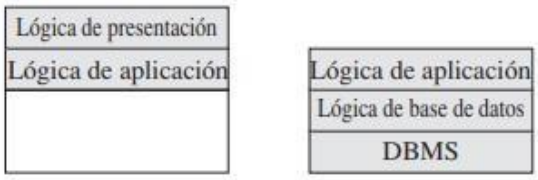


Figura 05. Procesamiento basado en el cliente.

- Procesamiento cooperativo: En este tipo de configuraciones, el procesamiento de la aplicación se realiza de forma óptima, beneficiándose de las maquinas cliente y servidora y de la distribución de los datos. Es más compleja de configurar y mantener, pero proporciona mayor productividad a los usuarios y mayor eficiencia de red que otras configuraciones cliente/servidor **[2].**

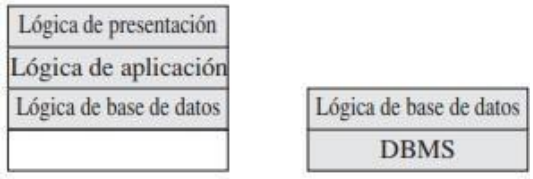


Figura 06. Procesamiento cooperativo.

El procesamiento basado en el cliente y el procesamiento cooperativo se corresponden con las configuraciones en que gran parte de la carga está en la parte cliente. **Las figuras 5 y 6** representan este modelo denominado cliente pesado (fat client). La principal ventaja de este modelo es que se beneficia de la potencia de los escritorios, descargando a los servidores y haciéndolos más eficientes y menos propensos a ser el cuello de botella. Existen, sin embargo, varias desventajas en la estrategia de los clientes pesados. Añadir nuevas funcionalidades suele sobrecargar la capacidad de los ordenadores de escritorio,forzando a las compañías a actualizarse.
Por último, es difícil mantener, actualizar o reemplazar aplicaciones distribuidas entre decenas o centenas de ordenadores. El procesamiento basado en servidor es

representativo de un enfoque de cliente ligero. Este enfoque imita al enfoque tradicional centralizado del host y es, a menudo, la ruta de migración para pasar las aplicaciones corporativas de los mainframes a un entorno distribuido.

## 2.2 – PASO DE MENSAJES DISTRIBUIDO [3]

En los sistemas de procesamiento distribuido, las computadoras generalmente no comparten la memoria principal, ya que cada una opera como un sistema independiente. Por lo tanto, las técnicas de comunicación entre procesos que dependen de la memoria compartida, como los semáforos, no son aplicables. En su lugar, se emplean métodos basados en el intercambio de mensajes. En esta y la próxima sección, exploraremos las dos técnicas más comunes. La primera es una implementación directa del uso de mensajes dentro de un único sistema. La segunda se basa también en el envío de mensajes, pero a través de llamadas a procedimientos remotos.

- La primera es una aplicación directa de los mensajes tal y como se utilizan en un único sistema.

▪ Un **“Cliente”** necesita algún servicio envía un mensaje con la petición de servicio a un proceso **“Servidor”**. El “Servidor” realiza la petición y envía un mensaje con la respuesta. En su forma más básica, sólo se necesitan dos funciones:

→ **“Send”** (mandar): La función Send especifica un destinatario e incluye el contenido del mensaje.

→ **“Receive”** (recibir). La función Receive indica de quién se desea recibir un mensaje y proporciona un buffer donde se almacenará el mensaje entrante.

- La segunda es una técnica que se basa en el paso de mensajes: las llamadas a procedimiento remoto.

▪ Los procesos hacen uso de los servicios de un módulo de paso de mensajes. Las peticiones de servicio se pueden expresar en términos de primitivas y parámetros.

→ Una primitiva especifica la función que se desea realizar, el formato real de las primitivas depende del software de paso de mensajes que se utilice, esta puede ser una llamada a procedimiento o puede ser un mensaje en sí mismo a un proceso que sea parte del sistema operativo.

→ Los parámetros se utilizan para pasar los datos y la información de control.

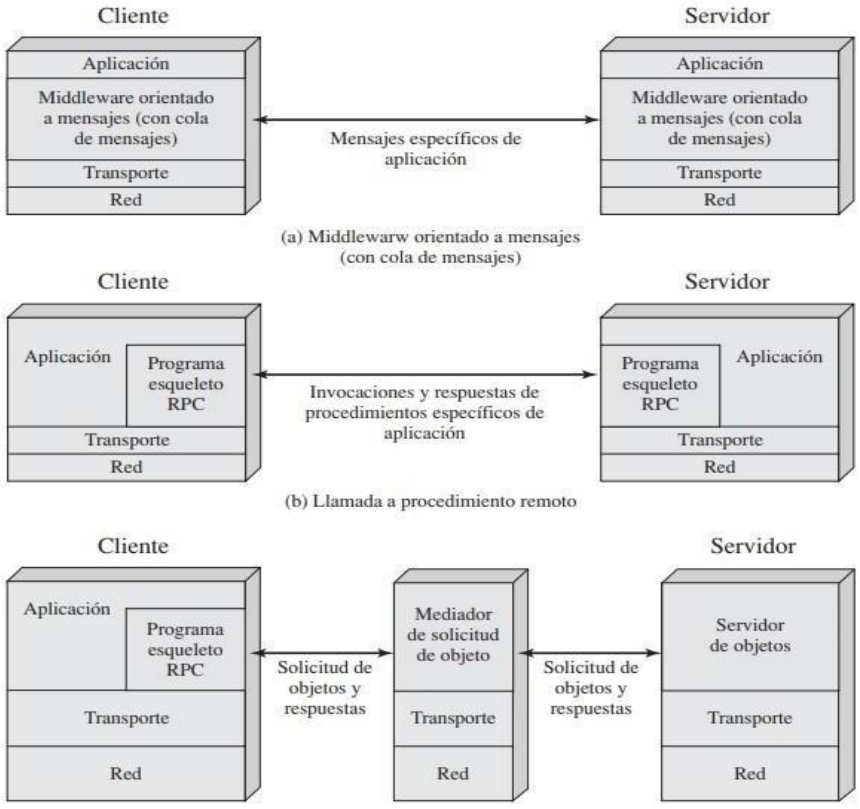


Figura 07. Mecanismo middleware


Figura 08. Primitivas básicas de paso de mensajes

### 2.2.1 – Fiable vs. No fiable

Un servicio fiable de paso de mensajes es el que garantiza la entrega si es posible. Estos servicios hacen uso de un protocolo de transporte fiable o de una lógica similar, y realizan comprobación de errores, acuse de recibo, retransmisión y reordenamiento de mensajes desordenados. Ya que se garantiza el envío, no es necesario informar al proceso emisor de que el mensaje ha sido enviado. En cualquier caso, si falla el envío del mensaje (por ejemplo, fallo persistente de la red, fallo del sistema de destino), se informa del fracaso al proceso emisor.

En el otro extremo, el servicio de paso de mensajes no fiable simplemente envía el mensaje por la red, pero no se le indica ni el éxito ni el fracaso. Esta alternativa reduce enormemente la complejidad y la sobrecarga de procesamiento y comunicación del servicio de paso de mensajes. **[9]**

### 2.2.2 - BLOQUEANTE VS. NO BLOQUEANTE

Con las primitivas no bloqueantes o asíncronas, no se suspende a un proceso como resultado de realizar un Send o Receive. De esta forma, cuando un proceso realiza una primitiva Send, el sistema operativo devuelve el control al proceso tan pronto como el mensaje ha sido puesto en la cola para su transmisión o se ha realizado una copia. Si no se realiza copia, cualquier cambio que el proceso emisor haga al mensaje, antes o durante su transmisión, se realizan bajo la propia responsabilidad del proceso. Cuando

el mensaje ha sido transmitido o copiado a un lugar seguro para su transmisión, se interrumpe al proceso emisor para informarle de que el buffer puede ser reutilizado. De forma similar, un Receive no bloqueante permite que el proceso continúe ejecutando. Cuando el mensaje llega, se informa al proceso con una interrupción, o bien el propio proceso puede verificar el estado periódicamente.

La alternativa es utilizar primitivas bloqueantes o asíncronas. Un Send bloqueante no devuelve el control al proceso emisor hasta que el mensaje ha sido transmitido (servicio no fiable) o hasta que el mensaje ha sido enviado y se ha recibido el acuse de recibo (servicio fiable). Un Receive bloqueante no devuelve el control hasta que el mensaje ha sido situado en su correspondiente buffer.

## 2.3 – LLAMADAS A PROCEDIMIENTO REMOTO

Las llamadas a procedimiento remoto son una variante del modelo básico de paso de mensajes. Hoy en día este método es muy común y está ampliamente aceptado para encapsular la comunicación en un sistema distribuido. Lo esencial en esta técnica es permitir a los programas en diferentes máquinas interactuar a través del uso de llamadas a procedimiento, tal y como lo harían dos programas que están en la misma máquina. Es decir, se utilizan las llamadas a procedimiento para acceder a los servicios remotos. El mecanismo de llamadas a procedimiento remoto se puede ver como un refinamiento del paso de mensajes fiable y bloqueante. La Figura 7(b) muestra la arquitectura general, y la Figura 9 proporciona una vista más detallada. El programa llamante realiza una llamada a procedimiento normal con parámetros de su máquina. Por ejemplo:

CALL P (X, Y) donde:

P = nombre del procedimiento

X = argumentos pasados

Y = valores devueltos

La intención de invocar a un procedimiento remoto en otra máquina puede ser transparente o no al usuario. En el espacio de direcciones del llamante se debe incluir el esqueleto (stub) de un procedimiento P o se debe enlazar dinámicamente en tiempo de llamada. Este procedimiento crea un mensaje que identifica al

procedimiento que se está llamando e incluye sus parámetros. De esta forma se envía el mensaje al sistema remoto y se espera una respuesta. Cuando se recibe una respuesta, el procedimiento esqueleto vuelve al programa llamante, proporcionando los valores devueltos.

En la máquina remota, hay otro programa esqueleto asociado con el procedimiento llamado. Cuando llega un mensaje, se examina y se genera un CALL P (X, Y) local. Este procedimiento remoto se llama localmente, de forma que las suposiciones normales de dónde encontrar los parámetros, el estado de la pila y otros, son idénticos al caso de una llamada local.

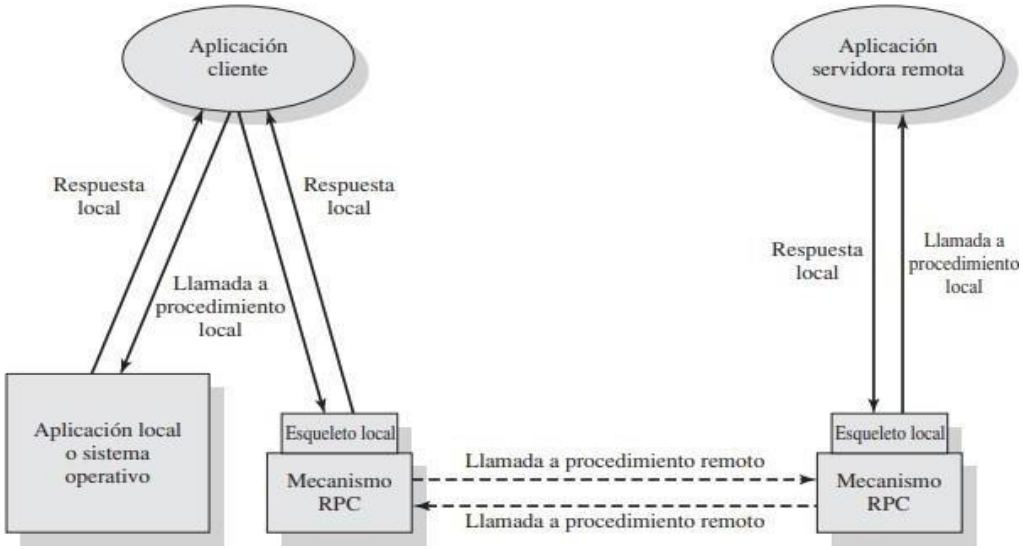


Figura 09. Mecanismo de llamadas a procedimiento remoto

### 2.3.1 - PASO DE PARÁMETROS

La mayor parte de los lenguajes de programación permiten que los parámetros se pasen como valores (denominado por valor) o como punteros que contienen los valores (llamado por referencia). Las llamadas por valor son fáciles de implementar en las llamadas a procedimiento remoto: los parámetros se copian en el mensaje y se envían al sistema remoto. Es más difícil implementar las llamadas por referencia. Para cada objeto se necesita un puntero único y válido para todo el sistema. No suele merecer la pena el esfuerzo por la sobrecarga que se genera **[4].**

### 2.3.2 – ENLACE CLIENTE/SERVIDOR

El enlace describe cómo se establecerá la relación entre un procedimiento remoto y el programa que lo invoca. Un enlace se crea cuando dos aplicaciones han establecido una conexión lógica y están listas para intercambiar datos y comandos.

Un enlace temporal implica que la conexión lógica entre los dos procesos se crea en el momento de la llamada al procedimiento remoto y se cierra tan pronto como se devuelven los resultados. Esta conexión requiere mantener información de estado en ambos extremos, lo que consume recursos, por lo que el estilo no permanente se utiliza para ahorrar estos recursos. Sin embargo, debido a la sobrecarga de establecer conexiones repetidamente, este tipo de enlace no es adecuado para procedimientos remotos que se llaman con frecuencia.

En cambio, los enlaces persistentes mantienen la conexión después de la ejecución de la llamada al procedimiento remoto y pueden ser reutilizados para futuras llamadas. Si no hay actividad en la conexión durante un periodo determinado, esta se cierra. Para aplicaciones que requieren invocaciones frecuentes de procedimientos remotos, el enlace permanente conserva la conexión lógica, permitiendo que varias llamadas consecutivas utilicen la misma conexión.

### 2.3.3 - SÍNCRONOS VS. ASÍNCRONOS

Las llamadas a procedimiento remoto (RPC) tradicionales son de tipo síncrono, lo que significa que el proceso que las invoca debe esperar a que el proceso al que se llama le devuelva un valor. En este sentido, el RPC síncrono se asemeja a una llamada a una subrutina.

El RPC síncrono es sencillo de comprender y programar debido a su comportamiento predecible. Sin embargo, no aprovecha el paralelismo inherente en los sistemas

distribuidos, lo que limita las interacciones de las aplicaciones distribuidas y puede resultar en un rendimiento más bajo.

Para aumentar la flexibilidad, se han desarrollado servicios de RPC asíncronos que permiten un mayor grado de paralelismo sin perder la simplicidad del RPC. En este modelo, las llamadas no bloquean al proceso que las invoca, lo que significa que las respuestas pueden recibirse cuando sea necesario, permitiendo que el cliente y el servidor trabajen en paralelo.
Un caso común de uso del RPC asíncrono es cuando un cliente realiza múltiples invocaciones a un servidor, manteniendo varias respuestas en proceso simultáneamente, cada una con su propio conjunto de datos, optimizando así la ejecución.

### 2.3.4 - MECANISMOS ORIENTADOS A OBJETOS

A medida que la tecnología orientada a objetos se vuelve más común en el diseño de los sistemas operativos, los diseñadores de la tecnología cliente/servidor han empezado a adoptar este enfoque. En este enfoque, los clientes y los servidores mandan mensajes entre objetos. La comunicación entre objetos se puede basar en una estructura de mensajes o RPC subyacente o puede estar directamente desarrollada sobre los servicios de orientación a objetos del sistema operativo.

Un cliente que necesita un servicio, manda una petición a un mediador de solicitud de objeto (object request broker), que actúa como un directorio de todos los servicios remotos disponibles en la red (Figura 14.10c). El mediador llama al objeto apropiado y le transfiere todos los datos relevantes. A continuación, el objeto remoto atiende la petición y responde al mediador, que devuelve la información al cliente.

## 2.4 - CLUSTERS

Los *Clusters* son una alternativa al Multiprocesamiento Simétrico, son sistemas que proporcionan un alto rendimiento y una alta disponibilidad y que son particularmente atractivos para aplicaciones de servidor **[7]**

Se enumera cuatro beneficios que se pueden lograr con los *cluster*. Estos beneficios también se pueden ver como objetivos o requisitos de diseño: **[5].**

- Escalabilidad absoluta: Es posible crear un gran *cluster* que supere la potencia de incluso la mayor de las máquinas. Un *cluster* puede tener decenas o incluso centenas de máquinas, cada una de ellas un multiprocesador.
- Escalabilidad incremental: Un *cluster* se configura de tal manera que sea posible añadir nuevos sistemas al *cluster* en pequeños incrementos. De esta forma, un usuario puede comenzar con un sistema pequeño y expandirlo según sus necesidades, sin tener que hacer grandes actualizaciones en las que un pequeño sistema debe ser reemplazado por uno mayor.
- Alta disponibilidad: Ya que cada nodo del *cluster* es una computadora en si´ mismo, el fallo de uno de los nodos no significa perdida del servicio. En muchos productos, el software maneja automáticamente la tolerancia a fallos.
- Relación precio/prestaciones: A través del uso de bloques de construcción es posible hacer un *cluster* con igual o mayor poder computacional que una única máquina mayor, con mucho menor coste.

### 2.4.1 - CONFIGURACIONES DE LOS CLUSTERS

Uno de los principales desafíos al construir un clúster es identificar y eliminar los puntos de fallo únicos (single points of failure).

Por ejemplo, en un clúster de supercomputación que depende de un servidor central para distribuir las tareas, si ese servidor falla, todo el clúster quedará inoperativo. De manera similar, en un clúster diseñado para balanceo de carga o alta disponibilidad, es crucial asegurar que los servidores continúen funcionando. Sin embargo, si estos servidores están conectados a una red corporativa o a Internet a través de una única interfaz, un fallo en esa interfaz podría aislar todo el sistema.

La redundancia es fundamental para prevenir que la falla de un solo componente de hardware —considerando que un clúster está compuesto por numerosos elementos, aumentando la probabilidad de fallos— afecte la funcionalidad del sistema en su totalidad.

Además, es esencial elegir la tecnología adecuada según nuestras necesidades. Por ejemplo, mantener un clúster en una red Ethernet de 10 Mb puede ser una buena opción

si hay pocos nodos, pero al agregar más, la red puede convertirse en un cuello de botella, haciendo que los servidores queden inactivos mientras esperan los datos durante períodos prolongados.

Los clústeres se pueden clasificar de diversas maneras, siendo la más simple aquella que se basa en si las computadoras del clúster comparten acceso a los discos.

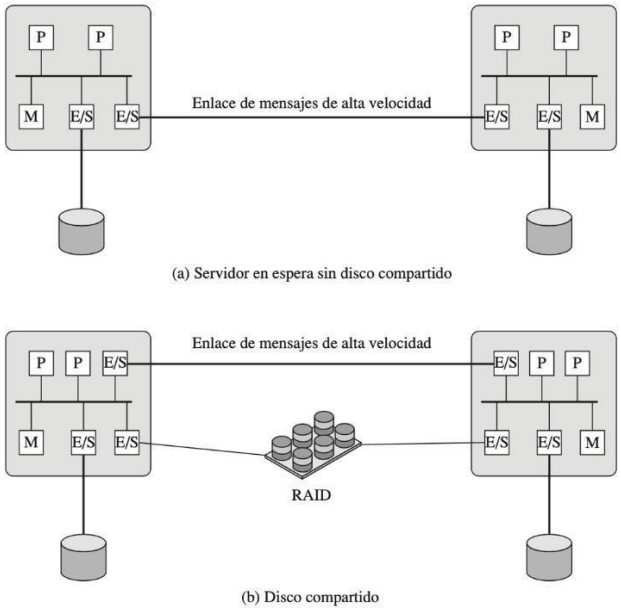


Figura 10. Configuraciones de cluster.

## 2.4.2 - ASPECTOS DE DISEÑO DE SISTEMAS OPERATIVOS

Para obtener todas las ventajas de una configuración hardware de un *cluster* se necesitan algunas mejoras en los sistemas operativos.

Gestión de Fallos. En general, para el tratamiento de los fallos se pueden seguir dos enfoques: *clusters* con alta disponibilidad y *clusters* con tolerancia a fallos.

Un *cluster* de alta disponibilidad ofrece alta posibilidad de que todos los recursos estén en servicio. Si sucede algún fallo, tal como la caída de un nodo o que se pierda un volumen, se pierden las peticiones en progreso. Cualquier petición perdida que se reintente, será atendida por una computadora diferente del *cluster*. Sin embargo, el

sistema operativo del *cluster* no garantiza el estado de las transacciones parcialmente realizadas. Esto se necesita gestionar a nivel de aplicación.

Un *cluster* tolerante a fallos asegura que todos los recursos están siempre disponibles. Esto se logra a través del uso de discos redundantes compartidos y mecanismos para deshacer transacciones incompletas.

La función de intercambiar una aplicación y los datos de un sistema fallido por un sistema alter-nativo del *cluster* se denomina recuperación de fallos (*failover*). La restauración de aplicaciones y de datos al sistema original una vez que se ha reparado, se denomina restauración de fallos (*failback*). La restauración puede ser automática, pero esto sólo es deseable si el problema está realmente solucionado y es poco probable que vuelva a suceder. De otra forma, la restauración automática puede provocar fallos sucesivos en el sistema inicial, generando problemas de rendimiento y de recuperación.

Equilibrado de carga. Un *cluster* necesita tener la capacidad de equilibrar la carga entre todas las computadoras disponibles. Esto incluye el requisito de que un *cluster* debe ser escalable. Cuando se añade una nueva computadora al *cluster*, el servicio de equilibrado de carga debe incluir automáticamente la nueva computadora en la planificación de las aplicaciones. Los mecanismos del *middleware* deben saber que pueden aparecer nuevos servicios en diferentes miembros del *cluster*, pudiendo migrar de un miembro a otro.

Computación paralela. En algunos casos, el uso eficiente de los *cluster* necesita ejecutar una única aplicación en paralelo.

Compilación paralela**.** Un compilador paralelo determina, en tiempo de compilación, qué partes de la aplicación se pueden ejecutar en paralelo. Estas partes se pueden asignar a computadoras diferentes del *cluster*. El rendimiento depende de la naturaleza del problema y de lo bueno que sea el diseño del compilador.

- Aplicaciones paralelas**.** En este enfoque, el programador escribe la aplicación para que ejecute en un *cluster* y utiliza paso de mensajes para mover datos, según se requiera, entre nodos del *cluster*. Esto supone una gran carga para el programador, pero probablemente es la mejor forma de explotar los *cluster* para algunas aplicaciones.

• Computación paramétrica**.** Este enfoque se puede utilizar si la esencia de la aplicación es un algoritmo que se debe ejecutar un gran número de veces, cada vez con un conjunto diferente de condiciones o parámetros iniciales. Un buen ejemplo es un modelo de simulación, que ejecutará un gran número de diferentes escenarios y luego calculará estadísticas de los resultados. Para que este enfoque sea efectivo, se necesitan herramientas de procesamiento paramétricas para organizar, ejecutar y gestionar los trabajos de forma ordenada.

### 2.4.3 – ARQUITECTURA DE UN CLÚSTER

Cada computadora es capaz de operar independientemente, además, en cada computadora está instalada una capa de software middleware que permite la operación del *cluster*. El middleware del *cluster* proporciona una imagen única al usuario, conocida como **imagen única del sistema** (*single-system image*). El middleware también podría´ ser responsable de proporcionar alta disponibilidad, a través de balanceado de carga y de la respuesta a los fallos de los componentes. Enumera los siguientes servicios y funciones deseables en un *cluster*: **[8]**

- **Un único punto de entrada.** Un usuario se autentifica en el *cluster* y no en una determinada computadora.
- **Una única jerarquía de ficheros.** Los usuarios ven una sola jerarquía de directorios bajo el mismo directorio raíz.
- **Un único punto de control.** Hay un nodo por defecto encargado de gestionar y controlar el *cluster*.
- **Una única red virtual.** Cualquier nodo puede acceder a cualquier otro punto del *cluster*, incluso si la configuración del *cluster* tienen múltiples redes interconectadas. Se opera sobre una única red virtual.
- **Un único espacio de memoria.** La memoria compartida distribuida permite a los programas compartir variables.
- **Un único sistema de control de trabajos.** Con un planificador de trabajos en el *cluster*, un usuario puede enviar su trabajo sin especificar la computadora que lo ejecutará.

- **Una única interfaz de usuario.** Todos los usuarios tienen un interfaz grafico común, independientemente de la estación de trabajo que utilicen.
- **Un único espacio de E/S.** Cualquier nodo puede acceder remotamente a cualquier periférico de E/S o disco, sin conocer su localización física.
- **Un único espacio de procesos.** Se utiliza un esquema uniforme de identificación de procesos. Un proceso en cualquier nodo puede crear o se puede comunicar con cualquier otro proceso en un nodo remoto.
- **Puntos de control.** Esta función salva periódicamente el estado del proceso y los resultados de computación intermedios, para permitir recuperarse después de un fallo.
- **Migración de procesos.** Esta función permite el balanceo de carga.

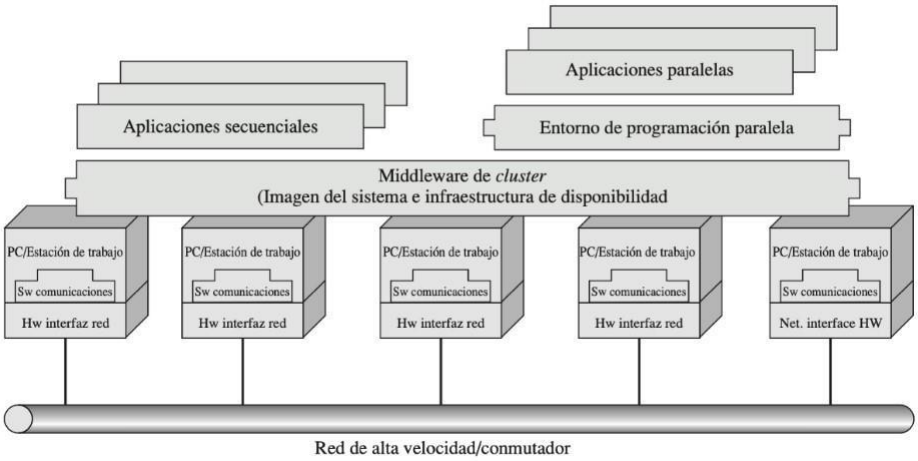


Figura 11. Arquitectura de computación cluster (BUY99a)

### 2.4.4 - *CLUSTER* FRENTE A SMP

Tanto los *clusters* como el multiprocesamiento simétrico proporcionan una configuración con múltiples procesadores para dar soporte a aplicaciones con alta demanda. Ambas soluciones están disponibles en el mercado, aunque SMP ha estado presente durante más tiempo.

La principal fuerza del enfoque SMP es que es más fácil de gestionar y configurar que un *cluster*. El SMP está mucho más cercano al modelo original de un solo procesador para los que están escritas prácticamente todas las aplicaciones. El principal cambio requerido para pasar de un uniprocesador a un multiprocesador es la función de planificación. Otro beneficio del SMP es que normalmente ocupa menos espacio físico y gasta menos energía que un´ *cluster* comparable. Por último, un beneficio importante es que los productos SMP están bien establecidos y son muy estables.

A largo plazo, sin embargo, las ventajas del *cluster* probablemente le llevarán a dominar el mercado de servidores de alto rendimiento. Los *clusters* son mucho más superiores que SMP en relación a la escalabilidad incremental y absoluta. Los *clusters* son también superiores en términos de disponibilidad, porque todos los componentes del sistema pueden ser altamente redundantes.

## 2.5 – SERVIDOR CLUSTER DE WINDOWS

El Servidor *Cluster* de Windows es un *cluster* no compartido, en que cada volumen de disco y otros recursos son propiedad de un único sistema a la vez.

El diseño del Servidor hace uso de los siguientes conceptos **[6].**

- Servicio *Cluster***.** Colección de software de cada nodo que gestiona actividades específicas del *cluster*.
- Recurso**.** Elemento gestionado por el servicio *cluster*. Son objetos que representan recursos reales en el sistema, incluyendo dispositivos hardware tales como discos o tarjetas de red y elementos lógicos tales como volúmenes lógicos de disco, direcciones TCP/IP etc.
- En línea (*online*). Un recurso está en línea en un nodo cuando está proporcionando servicio en ese nodo específico.

- Grupo**.** Colección de recursos gestionada como una unidad. Contiene todos los elementos necesarios para ejecutar una aplicación y para que los sistemas cliente se conecten al servicio proporcionado por esta aplicación.

Un grupo combina recursos en unidades mayores, tanto para la recuperación de fallos como para el balanceado de carga. Las operaciones realizadas en un grupo, tales como transferir el grupo a otro nodo, afectan a todos los recursos de ese grupo. Se implementan como bibliotecas dinámicas (DLL) y se gestionan por un monitor de recursos. El monitor de recursos interactúa con el servicio *cluster* a través de llamadas a procedimiento remoto y responde a los comandos del servicio *cluster* para configurar y mover grupos de recursos.

El **gestor de nodo** es responsable de mantener la pertenencia de este nodo al *cluster*. Periódicamente, manda mensajes a los gestores de nodo del resto. Cuando que un gestor de nodo detecte la pérdida de mensajes, difunde un mensaje a todo el *cluster*, haciendo que todos los miembros intercambien mensajes para verificar su estado. Si uno no responde, se le quita del *cluster* y sus grupos activos se transfieren a uno o más nodos activos del *cluster*.

El **gestor de la base de datos de configuración** guarda la base de datos de configuración del *cluster*. Contiene información de recursos, grupos, y pertenencia de grupos a nodos. Los gestores de base de datos de cada uno de los nodos del *cluster* cooperan para mantener la información de configuración. Para asegurar que los cambios, se utiliza software de transacciones tolerante a fallos.

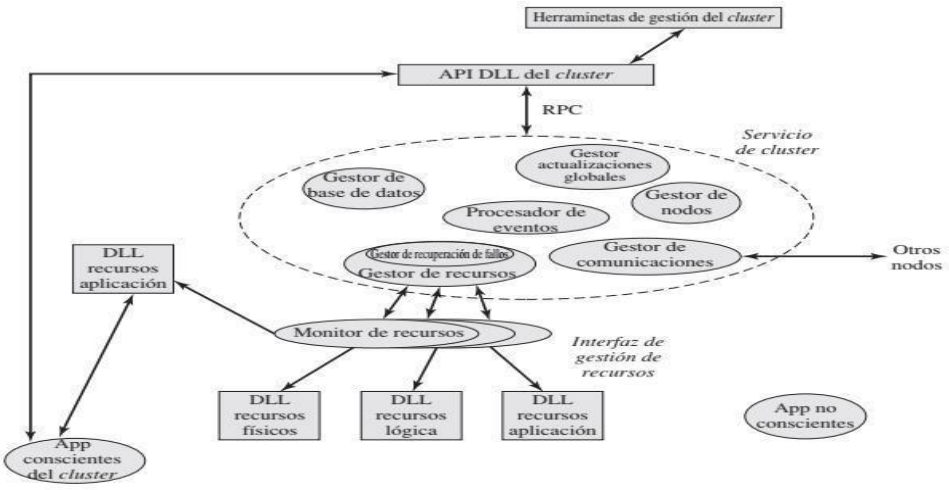


Figura 12. Diagrama de bloques del Windows Cluster Server (SHO97).

El gestor de recursos/gestor de recuperación de fallos toma las decisiones a los grupos de recursos e inicia acciones apropiadas tales como inicializar, reinicializar y recuperar fallos.

Cuando se requiere recuperar un fallo, el gestor de recuperación coopera para negociar una distribución de los grupos de recursos que ha fallado en el resto de los sistemas activos. Cuando un sistema se reinicia, el gestor de recuperación de fallos puede decidir hacer regresar algunos grupos a este sistema. Se puede configurar cualquier grupo con un propietario. Si ese propietario falla y luego se reinicia, el grupo se vuelve a llevar al nodo.

El procesador de eventos conecta todos los componentes del servicio, maneja operaciones comunes y controla la inicialización. El gestor de comunicaciones gestiona el intercambio de mensajes con el resto de los nodos. El gestor global de actualizaciones proporciona un servicio utilizado por otros componentes del cluster.

## 2.6 - SUN CLUSTER

Sun Cluster es un sistema operativo distribuido, construido como un conjunto de extensiones del sistema UNIX Solaris. Proporciona que, los usuarios y las aplicaciones ven al cluster como una única computadora ejecutando el sistema operativo Solaris.

Los principales componentes son:

- Soporte de objetos y comunicaciones.
- Gestión de procesos.
- Redes.
- Sistema de ficheros distribuido global.

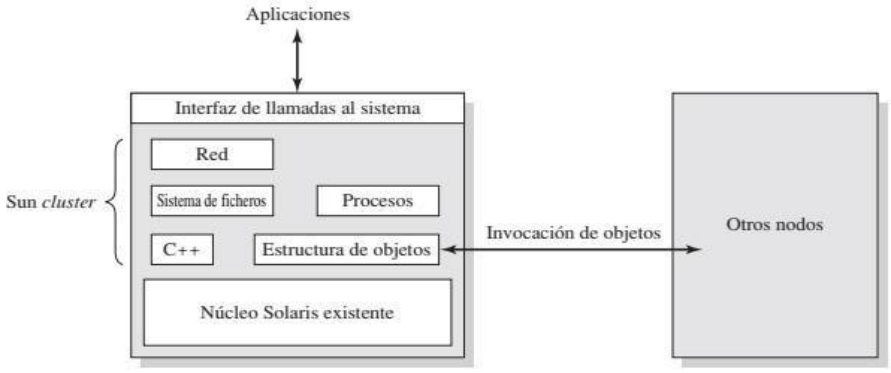


Figura 13. Estructura de Cluster

### 2.6.1 - SOPORTE DE OBJETOS Y COMUNICACIONES

La implementación está orientada a objetos. Se utiliza el modelo de objetos de CORBA para definir los objetos y las llamadas a procedimiento remoto (RPC). Se utiliza el Lenguaje de Definición de Interfaces (IDL) de CORBA para especificar los interfaces entre los componentes MC de los diferentes nodos. Los objetos de MC se implementan en C++. El uso de un modelo de objetos y de IDL, proporciona una comunicación entre los procesos, tanto dentro de un nodo como entre ellos.

### 2.6.2 - GESTIÓN DE PROCESOS

Extiende las operaciones de los procesos de forma que la localización de un proceso es transparente al usuario. Sun Cluster mantiene una visión global de los procesos de forma que en el cluster hay un identificador único por proceso y que cada nodo puede saber la localización y estado de cada proceso. Un proceso se puede mover de un nodo a otro, con el objetivo de lograr un balanceado de carga y de recuperar fallos. Sin embargo, todos los hilos de un proceso deben estar en el mismo nodo.

### 2.6.3 – REDES

Los diseñadores de Sun Cluster consideraron tres enfoques para el manejo del tráfico de red:

1. Realizar todo el procesamiento del protocolo de red en un único nodo. Para una aplicación en TCP/IP, el tráfico entrante podría ir a través de un nodo. Para el tráfico entrante el nodo analizaría las cabeceras TCP/IP y mandaría los datos encapsulados al nodo apropiado. Para el tráfico saliente, el nodo encapsularía los datos de otros nodos con cabeceras TCP/IP.
2. Asignar una dirección única y ejecutar los protocolos de red en cada nodo. Un problema es que la configuración del cluster deja de ser transparente. Otra es la dificultad de la recuperación de fallos cuando una aplicación en ejecución se mueve a otro nodo con una dirección de red diferente.
3. Utilizar un filtro de paquetes para enviar los paquetes al nodo apropiado y realizar el procesamiento del protocolo en dicho nodo. Las conexiones entrantes (peticiones de clientes), se balancean entre todos los nodos disponibles del cluster.

El subsistema de red de Sun Cluster tiene tres elementos principales:

i. Los paquetes entrantes se reciben en el nodo receptor que filtra el paquete y lo envía al nodo objetivo correcto usando la interconexión del cluster.
ii. Todos los paquetes salientes se envían a través de la interconexión del cluster al que tiene la conexión física de red externa. Todo el procesamiento del protocolo de los paquetes salientes se realiza en el nodo origen.
iii. Se mantiene una base de datos de configuración de red para anotar el tráfico de red de cada nodo.

### 2.6.4 - SISTEMA DE FICHEROS GLOBAL

El elemento más importante de Sun Cluster es el sistema de ficheros global, que compara la gestión de ficheros de MC con el esquema básico de Solaris. Ambos se basan en el uso de los conceptos nodo-v y sistema de ficheros virtual.

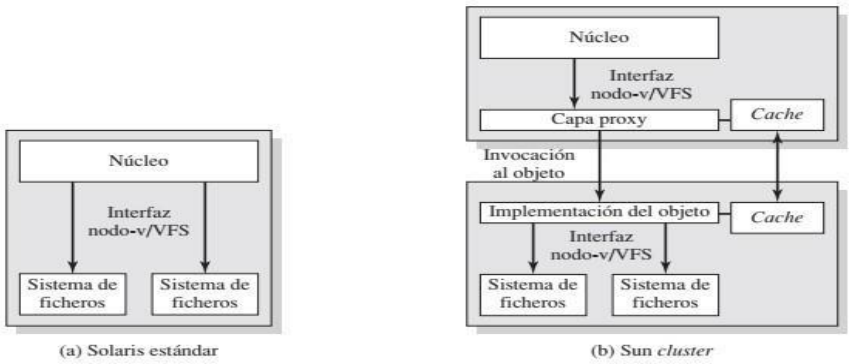


Figura 14. Extensiones del sistema de fichero de Sun Cluster.

En Solaris, la estructura de nodo virtual (nodo-v) se usa para ofrecer una interfaz común para todos los tipos de sistemas de archivos. El nodo-v asocia páginas de memoria con el espacio de direcciones de un proceso y autoriza el acceso a un sistema de archivos. A diferencia de los nodos-i, que vinculan procesos con archivos UNIX, los nodo-v pueden conectar un proceso con un objeto de cualquier sistema de archivos. Así, una llamada al sistema solo necesita saber cómo utilizar la interfaz del nodo-v, sin importar el objeto que maneje.

Esta interfaz acepta comandos como leer o escribir, y los convierte en acciones específicas del sistema de archivos. Los nodo-v describen objetos individuales, mientras que las estructuras de sistema de archivos virtual describen el sistema completo.

En Sun Cluster, el sistema de archivos global proporciona una interfaz unificada para archivos distribuidos en el cluster. Un proceso puede acceder a un archivo en cualquier parte del cluster usando la misma ruta. Para esto, se emplea un sistema de archivos proxy sobre el existente en Solaris usando la interfaz nodo-v, permitiendo que los objetos invocados residan en cualquier nodo. Este entorno no requiere modificar el núcleo ni el sistema de archivos base. Además, se utiliza una caché para reducir invocaciones remotas, almacenando contenido de archivos, información de directorios y atributos.

## 2.7 - CLUSTERS BEOWULF Y LINUX

En 1994 se inició el proyecto Beowulf con el patrocinio del proyecto de la NASA High Performance Computing and Communications (HPCC)

### 2.7.1 - CARACTERÍSTICAS DE BEOWULF

Las principales características de Beowulf incluyen las siguientes [RIDG97]:

- Componentes genéricos disponibles en el mercado.
- Procesadores dedicados (mejor que ciclos disponibles de estaciones de trabajo ociosas).
- Una red privada y dedicada (LAN o WAN o una combinación de redes).
- Ningún componente propio.
- Fácilmente replicable para múltiples vendedores.
- E/S escalable.
- Basado en software gratuito disponible.
- Utiliza herramientas de computación gratuitas con mínimos cambios.
- Retorno del diseño y de las mejoras a la comunidad.

Aunque los elementos software de Beowulf se han implementado en diversas plataformas, la elección más obvia se basa en Linux, y la mayor parte de las implementaciones Beowulf utiliza un cluster de estaciones de trabajo Linux sobre PCs. El cluster tiene una serie de estaciones de trabajo, posiblemente con diferentes plataformas hardware, ejecutando el sistema operativo Linux. El almacenamiento secundario de cada estación de trabajo puede estar disponible para acceso distribuido (para compartición de ficheros distribuida, memoria virtual distribuida y otros usos). Los nodos del cluster (los sistemas Linux) se interconectan con una red, normalmente Ethernet. La Ethernet puede estar basada en un solo conmutador (switch) o en un conjunto de conmutadores interconectados. Se utilizan productos Ethernet con velocidades estándar (10 Mbps, 100Mbps y 1 Gbps).

### 2.7.2 - BEOWULF SOFTWARE

El software Beowulf es una extensión de distribuciones Linux gratuitas. Cada nodo del cluster ejecuta su propio núcleo de Linux y puede funcionar de forma autónoma, pero se realizan modificaciones para permitir a los nodos compartir espacios de nombres globales.

Ejemplos:

- BPROC: Expande el espacio de procesos a múltiples nodos y permite iniciar procesos en nodos remotos, dando una visión unificada del cluster.
- Unión de canales Ethernet: Combina múltiples redes Ethernet en una sola, mejorando el ancho de banda y balanceando la carga.
- Pvmsync: Proporciona sincronización y datos compartidos para procesos en el cluster.
- EnFuzion: Ejecuta trabajos paramétricos con diferentes parámetros en el cluster.

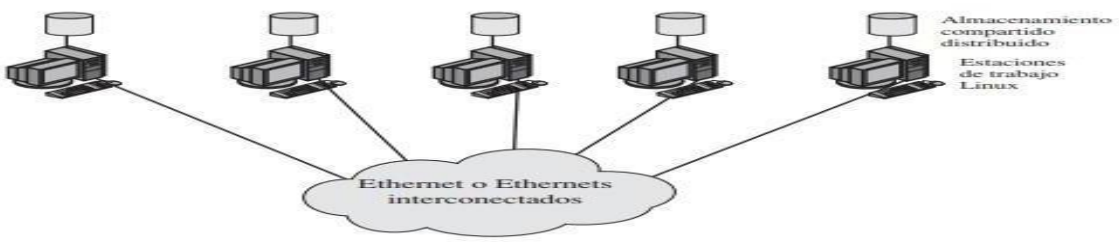


Figura 15. Configuración genérica de Beowulf.

## 3 – CONCLUSION

En resumen, la arquitectura cliente/servidor, junto con la computación distribuida y el uso de clústers, se ha convertido en un pilar fundamental para el desarrollo de sistemas de información eficientes y escalables.

La computación cliente/servidor es esencial para aprovechar al máximo los recursos de una red y mejorar la productividad organizacional, ya que distribuye las aplicaciones entre los clientes y los servidores, facilitando el acceso eficiente a recursos compartidos. Los clientes interactúan a través de interfaces gráficas sencillas, mientras que los servidores gestionan tareas clave, como bases de datos, dividiendo las responsabilidades para optimizar el rendimiento. En los sistemas distribuidos, la comunicación entre procesos es clave, utilizando técnicas como el paso de mensajes y las llamadas a procedimientos remotos para permitir la interacción entre programas en distintas máquinas como si estuvieran en la misma. Además, los clusters de computadoras unifican múltiples sistemas en una sola entidad, proporcionando un entorno de procesamiento más potente y flexible, combinando la descentralización con la gestión centralizada de los recursos.

## 4 – REFERENCIA BIBLIOGRAFICA


- **[1].** Oracle, "Overview of SQL Statements," *Oracle Database SQL Language Reference 11g Release 1 (11.1)*, https://docs.oracle.com/cd/B28359_01/server.111/b28286/intro002.htm#SQLRF50928.
- **[2]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 páginas 625, 626.
- **[3]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 página 630
- **[4]** Gibbons, P. «A Stub Generator for Multilanguage RPC in Heterogeneous Environments.» IEEE Transactions on Software Engineering. Enero 1987.
- **[5]** Brewer, E. «Clustering: Multiply and Conquer.» Data Communications, Julio1997.
- **[6]** Short, R.; Gamache, R.; Vert, J. y Massa, M. «Windows NT Clusters for Availability and Scalability.»
- **[7]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 página 636.

- **[8]** Hwang, K, et al. «Designing SSI Clusters with Hierarchical Checkpointing andSingle I/O Space.» IEEE Concurrency, enero-marzo 1999.
- **[9]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 página 632.