CAPÍTULO 3
La Distribución y los Sistemas Operativos

Matias Banchoff, Nicolás del Rio, Lía Molinari

y Juan P. Pérez

Quizá el lector sea un avezado conocedor de sistemas operativos. Quizás no. Lo que podemos afirmar es que seguramente está familiarizado con ellos. La masividad en el uso de computadoras y celulares hizo que aún los más neófitos, los recién llegados, hayan oído hablar de Windows, de Linux, de Android, etc.

El sistema operativo de una computadora surge injustamente como el primer culpable de cualquier mal que la aqueje. Que si es lento, que no es suficientemente amigable, que no soporta esta aplicación, que hay que actualizarlo. Pero… es imprescindible para poder interactuar con el hardware y ejecutar los programas.

Por eso, en un acto de reivindicación, vamos a dedicarle este capítulo en el contexto de los sistemas distribuidos y analizaremos su rol trascendental.

3.1. ¿Qué es un sistema operativo?

Hace muchos años para definir a los Sistemas Operativos era suficiente hacerlo como un administrador de recursos de hardware y software. La interacción con el usuario no era una variable a considerar. Quienes utilizaban las computadoras eran especialistas. La élite de los albores: matemáticos, físicos, ingenieros.

Hoy cuando definimos los sistemas operativos lo hacemos desde dos miradas: como administrador de recursos y la facilidad de uso. La masividad y la ubiquidad de las TICs exigieron que los desarrolladores de los sistemas operativos dieran a la amigabilidad un rol preponderante en las nuevas versiones.

En este libro, donde nos referimos a los sistemas distribuidos, cabe analizar cuál es el rol de los sistemas operativos en ellos, cuáles son los recursos a administrar, qué funciones debe llevar a cabo y las características que debe presentar ante la homogeneidad o heterogeneidad de los equipos conectados.

Un sistema distribuido es un conjunto de computadoras independientes que aparecen ante los usuarios como una única computadora. Las ventajas esperadas con respecto a velocidad,

confiabilidad, distribución, escalabilidad y extensibilidad no podrían lograrse si no existiera una entidad que administre los recursos y garantice su disponibilidad y sincronización.

La compartición de recursos es uno de los temas clave cuando se analiza el contexto de los sistemas distribuidos.

Ante estos escenarios donde interactúan varias CPU, una de las principales cuestiones es de qué forma lo hacen. Puede ser que tengan independencia e interactúen cuando lo precisen; que lo hagan mediante una red, o compartiendo memoria; que cada CPU tenga un rol definido o que entre varias lleven a cabo una tarea.

Este acoplamiento define el grado de integración.

El sistema operativo, como administrador de recursos, dará el soporte necesario de acuerdo a cómo se realiza esta integración.

Por lo tanto, una y otra vez en este libro aparecerá el término abstracción. Para soportar la heterogeneidad de recursos se desarrollan capas de software para integrarlos, que *abstraen* las características propias de estos recursos: redes, hardware, lenguajes de programación, e incluso, sistemas operativos.Una de esas capas de software es el middleware, sobre el cual nos referiremos en párrafos posteriores.

Podríamos incluso pensar en un sistema operativo como un software de abstracción. Por ejemplo, facilita la integración de diferentes dispositivos de almacenamiento o de comunicación. Gracias a él no necesitamos hablar de sectores de disco, o sockets. La masividad en el uso de la tecnología no hubiera sido posible sin la evolución de los sistemas operativos, ¿o es al revés?

Volvamos al concepto de pensar en un conjunto de computadoras interconectadas.

Puede ser que cada una de ellas tenga su propio sistema operativo y que puedan acceder a recursos remotos, por ejemplo, a un servidor de bases de datos.

En este esquema, cada nodo tiene autonomía en la gestión de sus recursos. Un usuario puede acceder a otro nodo y ejecutar allí un programa. Pero no hay planificación de procesos entre los distintos nodos.

Analicemos otra alternativa donde el usuario que quiere ejecutar un programa en otro nodo no tiene que conectarse a ese nodo, o no debe indicar dónde se ubica un recurso remoto, porque existe un nivel de transparencia que permite acceder a lo remoto casi como si fuera local.

Estas dos posibilidades son lo que se llama Sistemas operativos en red y sistemas operativos distribuidos.

El sistema operativo distribuido controla todos los nodos y puede enviar a ejecutar un programa en cualquier nodo de acuerdo a una política de planificación.

3.2. Hitos en la historia de los sistemas operativos

Cuando se describe la evolución de los sistemas operativos, es ineludible hablar de los mainframes. En la saga de libros de Sistemas Operativos de Avi Silberschatz, Peter Baer Gal-

vin y Greg Gagne, se utilizan los dinosaurios para representar la etapa de la informática donde los mainframes eran los reyes de la comunidad. Enormes, pesados, lentos pero sumamente seguros en su esquema fuertemente centralizado, aunaban software de base, ambientes de desarrollo y bases de datos del mismo proveedor, garantizando un cliente cautivo, obligado a adaptarse a cambios de versiones y consecuentes migraciones. La empresa proveedora tenía una fuerte injerencia en las decisiones tecnológicas de la organización para garantizar el servicio y la integración. El proveedor marcaba las tendencias.

Los modelos como los 360, 370, 390 de IBM, con sus sistemas operativos VM, VSE, MVS, Z/OS, o los provistos por Burroughs, Unisys u otros, o renovados soportando Linux, como Linux para Zseries en nuestros días. Mantienen sus adeptos entusiasmados por la posibilidad de procesamiento de transacciones a gran escala, el soporte de miles de usuarios y aplicaciones, el acceso simultáneo a los recursos, gran capacidad de almacenamiento, comunicaciones mediante gran ancho de banda. Y en forma muy destacada: seguridad, robustez.

A mediados de los 90, en el mercado argentino irrumpen los sistemas abiertos, como amenazante alternativa al monopolio de los mainframes.

El atributo de abierto surge de la posibilidad de combinar interoperabilidad, portabilidad y uso de computadoras de diferentes proveedores. Para ello se especifican interfaces, servicios y formatos.

Este nuevo contexto influyó en un cambio de capacidades y habilidades en el personal de administración de sistemas o soporte técnico. La integración de sistemas operativos, bases de datos, administración de redes y lenguajes revolucionó las hasta el momento bastante pasivas áreas de soporte, que sólo conocían los productos de un sólo proveedor, que además, definía el presente y el futuro de la infraestructura de la organización.

Esta etapa fue la que, sin intención, comenzó a preparar a los profesionales de TI en la revolución del software libre que la sucedió. Integración. Interoperabilidad. Colaboración.

El sistema operativo Unix deja de ser un producto para universidades y las grandes marcas revolucionan el mercado con AIX-IBM, HP-UX. La terminología que se utilizaba evidenciaba a qué bando se pertenecía: los de IBM *ipeleaban* (argentinismo que proviene de ejecutar el IPL, Initial Program Loader) y la gente de Unix, *booteaba* (argentinismo por ejecutar el proceso boot, de arranque).

3.3. Cliente-servidor: un importante paso hacia los sistemas distribuidos

Las arquitecturas cliente-servidor impusieron un modelo determinado por los servicios.

Este esquema es un modelo de aplicación distribuida, en el que las tareas o actividades se reparten entre un conjunto de procesos proveedores de recursos o servicios llamados servidores, y un conjunto de procesos demandantes de recursos o servicios llamados clientes. Si bien este esquema puede ser aplicado a procesos que se ejecuten en una misma computadora, el

mayor potencial se presenta al pensar a los mismos interactuando de manera remota sobre una red de computadoras.

Hoy en día podemos encontrar el modelo cliente/servidor en numerosas situaciones que utilizamos a diario: servicio de Correo Electrónico, acceso a la *World Wide Web*, aplicaciones en nuestros teléfonos móviles para compra de productos o servicios, servidores de bases de datos, entre muchas más.

En esta arquitectura la capacidad de proceso está distribuida entre los procesos clientes y los procesos servidores, lo que brinda una importante ventaja centralizando la gestión de la información y separando las responsabilidades en la organización de las aplicaciones.

La separación entre cliente y servidor es lógica. El servidor no necesariamente se ejecuta sobre una sola computadora ni exige la existencia de un único proceso. Esta característica permite mejorar los tiempos de respuesta.

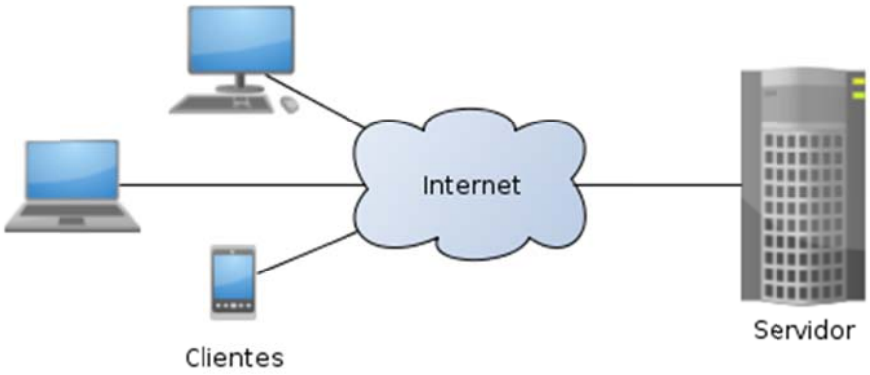

Fig. 25. Diferentes Clientes usando un Servidor

La arquitectura cliente servidor se basa en el modelo de comunicación requerimiento-respuesta (*request-reply protocol*). Esto es, la comunicación se basa en un mensaje del cliente al servidor (el requerimiento) y un mensaje del servidor al cliente (la respuesta). En este modelo, el cliente esperará desde que envía el mensaje hasta que recibe la respuesta al mismo.

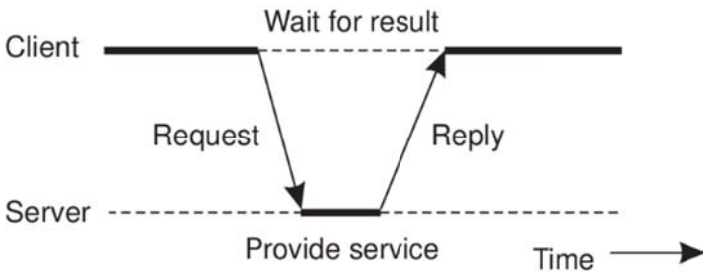

Fig. 26. Esquema del modelo de comunicación Requerimiento - Respuesta

Veamos algunas características de ambos tipos de procesos. Por un lado, los clientes:

- son los responsables de iniciar las solicitudes por lo que tienen un papel activo en la comunicación;

- esperan y reciben las respuestas del servidor;
- pueden conectarse a varios servidores a la vez para cumplir una tarea específica;
- normalmente interactúan con los usuarios finales mediante alguna interfaz gráfica de usuario o GUI (Graphical User Interface).
- En cuanto a los servidores:
- al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñando entonces un papel pasivo en la comunicación;
- ante la recepción de una solicitud, la procesan y luego envían la respuesta al cliente;
- pueden aceptar las conexiones de un gran número, aunque limitado, de clientes.

Comúnmente a la arquitectura tradicional cliente/servidor se la conoce como arquitectura de dos capas: una capa cliente y un servidor. En el siguiente gráfico podemos observar diferentes esquemas que se pueden presentar. Una alternativa plantea toda la lógica a cargo del servidor dejando únicamente la presentación de la información en el cliente. En otro caso el servidor sólo resulta responsable de los datos y el cliente lo es del resto de las actividades - presentación, gestión, etc., existiendo además esquemas alternativos entre los dos extremos mencionados.

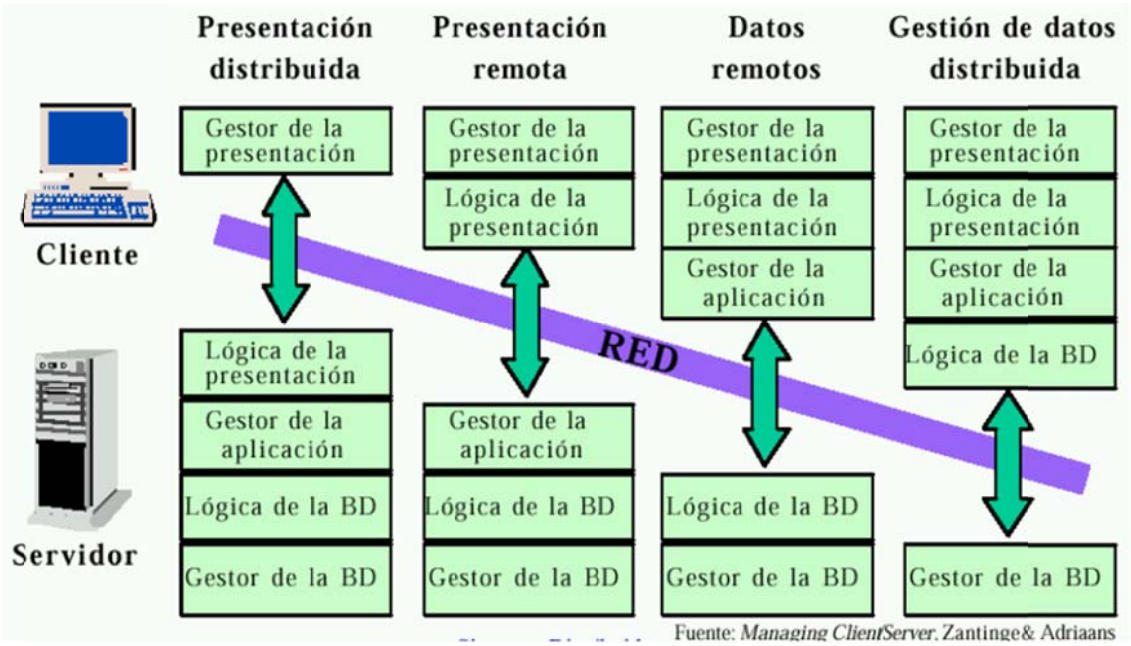

Fig. 27. Posibles organizaciones Cliente/Servidor en un modelo de Dos Capa

Otra posible organización es la conocida como de tres capas:

- Una capa cliente que interactúa con los usuarios finales.
- Una capa servidor de aplicación, la cual se encarga de procesar los datos para los clientes.
- Una capa servidor de base de datos, responsable de almacenar los datos y brindar los mismos a la capa servidor de aplicación.

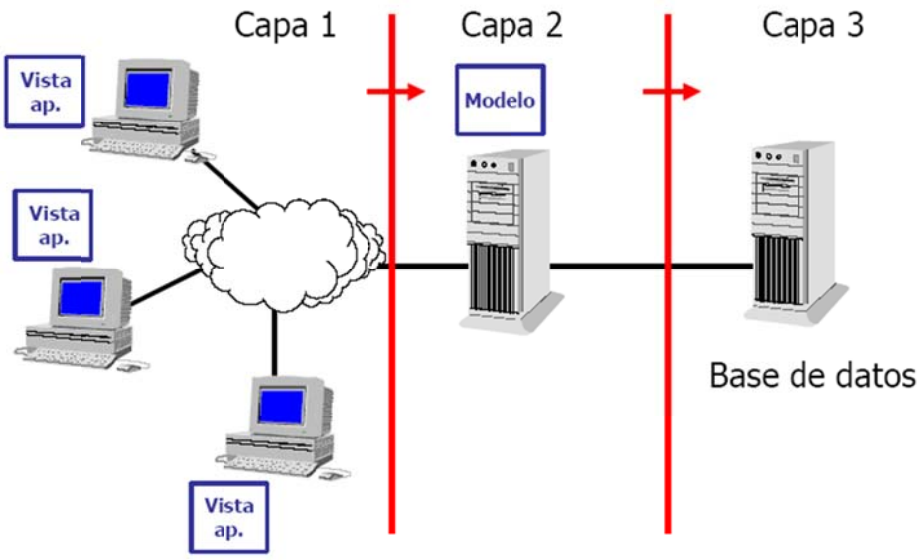

Fig. 28. Organización Cliente/Servidor de Tres Capas

La utilización de múltiples capas nos presenta la ventaja de obtener un mayor desacople en los pasos requeridos para dar una respuesta a un usuario, mejorando el balance de la carga de cada nodo participante y haciendo la solución más escalable. Las desventajas que se presentan las encontramos en la mayor carga producida en la red - por el aumento del tráfico de información - así como en la dificultad de desarrollar, probar y desplegar aplicaciones respecto a una solución de 2 o 3 capas.

De acuerdo con (Bochenski 1994), son diez las características que definen un ambiente cliente/servidor. La existencia de las cinco primeras es de carácter obligatorio, mientras que las otras cinco es de cumplimiento opcional:

1. Una arquitectura cliente/servidor consiste de un proceso cliente y un proceso servidor que pueden ser distinguidos uno del otro y que pueden interactuar de manera independiente.
2. Las partes cliente y servidor pueden operar, aunque no necesariamente, en plataformas computacionales diferentes.
3. Tanto la parte cliente como la del servidor pueden ser actualizadas individualmente sin que la otra deba serlo también.
4. El servidor es capaz de dar servicio a múltiples clientes en forma concurrente. En algunos sistemas pueden acceder múltiples servidores.
5. Un sistema cliente/servidor incluye algún tipo de capacidad de red.
6. Una porción significativa (a veces la totalidad) de la lógica de la aplicación reside en el cliente.
7. El procesamiento es iniciado usualmente en el lado del cliente, no del servidor. Sin embargo, los servidores de bases de datos pueden iniciar acciones basadas en “disparos automáticos”, “reglas del negocio” o procedimientos almacenados.
8. Una interfaz gráfica de usuario amigable generalmente reside en el lado del cliente.

9. La capacidad de un lenguaje estructurado de consultas es una característica de la mayoría de los sistemas cliente/servidor.
10. El servidor de base de datos debería proporcionar seguridad y protección a los datos.

Por su parte (Orfali 1994) resume así las características de los sistemas cliente/servidor:

- **Servicio**: El ambiente cliente/servidor conforma una relación entre procesos. El proceso servidor ofrece servicios, mientras que los clientes los solicitan.
- **Recursos compartidos:** Los servidores regulan el acceso a los recursos comunes por parte de los clientes.
- **Protocolos asimétricos:** Los clientes pueden pertenecer a una amplia variedad de tecnologías, y son los responsables por iniciar las solicitudes de servicios. Un servidor es un ente pasivo que se encuentra en espera permanente por dichas solicitudes.
- **Transparencia de localización:** Aun cuando un proceso servidor puede residir en el mismo equipo que los procesos clientes, es indispensable que los sistemas cliente/servidor oculten a éstos la localización física de los servidores, redireccionando apropiadamente las llamadas a los servicios requeridos.
- **Apertura:** El software cliente/servidor debe ser lo más independiente posible de las plataformas de hardware y sistemas operativos involucrados.
- **Intercambios basados en mensajes:** Los servidores y los clientes participan de sistemas *débilmente acoplados*, cuya relación se implementa con mecanismos de paso de mensajes (Message-Passing).
- **Encapsulación de servicios:** Los procesos servidores deben poder ser actualizados sin que esto provoque cambios en los clientes. Para lograr lo anterior, únicamente es necesario mantener sin alterar la interfaz de comunicación entre ambos componentes.
- **Escalabilidad:** El escalamiento de los sistemas cliente/servidor puede ser horizontal (adición o eliminación de clientes sin afectar significativamente el rendimiento global de un sistema) o vertical (crecimiento hacia configuraciones más grandes y eficientes).
- **Integridad:** Los servicios y datos de un servidor se ubican en un lugar centralizado, lo que simplifica su mantenimiento y protección.

Las principales ventajas que brinda la arquitectura cliente/servidor son:

- **Mantenibilidad:** la descomposición de sistemas rígidos y monolíticos hacia partes discretas intercomunicadas facilita el mantenimiento y reduce los costos. Como sucede con la mayoría de productos de la ingeniería, es más fácil dar servicio, reemplazar y arreglar componentes con interfaces bien definidas, que hacer el equivalente en unidades monolíticas.

- **Modularidad:** la arquitectura cliente/servidor está construida sobre la base de módulos conectables. Tanto el cliente como el servidor son módulos del sistema independientes uno del otro y pueden ser reemplazados sin afectarse mutuamente. Se agregan nuevas funciones al sistema ya sea creando nuevos módulos o mejorando los existentes.
- **Adaptabilidad:** el desacoplamiento del cliente y del servidor permite una rápida solución ante cambios del entorno del cliente, con modificaciones mínimas o nulas.
- **Escalabilidad:** las soluciones cliente/servidor pueden ser orientadas a satisfacer las necesidades cambiantes de la empresa.
- **Portabilidad:** actualmente el poder de procesamiento se puede encontrar en varios tamaños: super servidores, servidores, desktop, notebooks, máquinas portátiles. Las soluciones cliente/servidor basadas en estándares permiten a las aplicaciones estar localizadas donde resulte más ventajoso u oportuno.
- **Sistemas abiertos:** los sistemas cliente/servidor pueden desarrollarse bajo la premisa de sistemas basados en estándares de la industria.
- **Autonomía:** las máquinas cliente pueden presentar diversas configuraciones, tamaños, marcas y arquitecturas. Con una configuración adecuada, cada cliente puede trabajar en forma independiente o como parte de la red distribuida de la empresa.

Si bien vimos que esta arquitectura proporciona numerosas ventajas, no conforman la solución perfecta para las necesidades de administración de la información en una empresa, debido a que imponen también ciertas restricciones, entre las que encontramos (Price 1995):

- Si una parte importante de la lógica de las aplicaciones es trasladada al servidor, éste puede convertirse en un *cuello de botella* del sistema global. En este caso, los recursos limitados del servidor tienen una alta demanda por un número cada vez más creciente de usuarios.
- Las aplicaciones distribuidas, en especial aquellas basadas en un modelo cooperativo, son más complejas que las no distribuidas, e imponen cargas adicionales de comunicación y por ende de transferencia de información (datos del usuario y *overhead* del sistema).
- Se requiere de un alto grado de compatibilidad y de sujeción a estándares de parte de los dispositivos (hardware y software) que conforman un sistema cliente/servidor, para que éste pueda funcionar de una manera efectiva y transparente para el usuario.
- La mayoría de las herramientas cliente/servidor obligan a los usuarios a aprender esquemas de desarrollo de aplicaciones totalmente nuevos para muchos.
- La complejidad y el esfuerzo requerido para administrar y soportar un ambiente cliente/servidor grande basado en sistemas abiertos es un punto a considerar. Hay pocas herramientas bien reconocidas que soportan manejo de configuraciones, monitoreo del rendimiento y distribución de versiones de software.

Pero ¿qué pasa con los sistemas operativos en el caso que el cliente y el servidor funcionen en computadoras diferentes?

El sistema operativo de la máquina donde se ejecuta el cliente, posiblemente esté orientado a un uso pensado en el ámbito del consumidor de servicios, por ejemplo, un puesto de trabajo donde el usuario accede a servidores de correo, o a servidores de páginas web, o que usa aplicaciones cliente específicas de una aplicación determinada.

No obstante, hay sistemas operativos que permiten alojar un pequeño servidor web con algunas limitaciones.

Los sistemas operativos de las máquinas que ejecutan procesos servidores están orientadas a optimizar el servicio que se ofrece a los clientes. Estas computadoras habitualmente están siempre encendidas y tendrán características físicas orientadas a su servicio. Si es un servidor de archivos, por ejemplo, tendrá que tener capacidad de almacenamiento adecuada. Si es un servidor de una aplicación que se espera sea muy accedida, debe estar configurado y preparado para atender múltiples demandas concurrentes.

Si fuera accedido por el resto de las computadoras en una red, en este servidor se alojarían los servicios de dominio, de DNS, de DHCP, entre otros.

3.4. La nube y la grid

En párrafos anteriores se enunciaron los conceptos tradicionales en el marco de los sistemas operativos. Pero la evolución de la tecnología en las últimas décadas favoreció el surgimiento de nuevos modelos de computación distribuida. Es el caso de la computación en la nube (*cloud computing*) y computación en la *grid* (*grid computing*). La administración de recursos se hizo más compleja pues se agrega una arquitectura de software para lograr la abstracción que se pretende en la interacción con estos sistemas y el usuario.

El uso de *grid computing*, se inicia a principios de los 90s como respuesta a una necesidad planteada desde el mundo científico, facilitando la integración de recursos computacionales. Es una forma de computación distribuida que fue ganando adeptos en universidades, grandes centros de investigación y laboratorios.

Cloud computing, también llamada *computación en la nube*, es un paradigma que surge con el objetivo de ofrecer servicios de computación a través de Internet. Hasta aquí no queda muy claro cuál es la diferencia entre ambos modelos: al fin y al cabo, ambos ofrecen servicios por Internet.

Más allá de las diferencias planteadas por Foster et al, ambos son modelos de computación distribuida.

Un sistema operativo para la nube (OS *cloud computing*), también llamado sistema operativo virtual, está diseñado para administrar los recursos de este entorno, de acuerdo al modelo de prestación. La virtualización, de uso habitual en el contexto de la nube, también define

nuevas pautas para la creación, operación y mantenimiento de las máquinas virtuales, servidores e infraestructura.

Una forma de definir la *grid* es mediante un conjunto de *clusters* geográficamente dispersos comunicados por redes de alta velocidad y con una infraestructura de software para permitir la interacción e interoperación. Este software es llamado middleware.

¿Es el middleware un sistema operativo? Administra recursos y permite trabajar en forma remota con diferentes grados de abstracción. De acuerdo a la definición tradicional de sistemas operativos, podría entrar en esa categoría. Pero este middleware es complejo de configurar, exigiendo un importante grado de especialización, difícil de conseguir en ambientes no científicos. Además está orientado a administrar un subconjunto de recursos, no todos los recursos.

Sea el contexto que planteemos, la definición más amplia acerca de qué es un sistema operativos, lo generalizamos aceptando que es el software que administra los recursos de una manera eficiente y facilita su acceso con un alto grado de transparencia.

3.5. Una introducción a la virtualización

Virtualizar es un término que se ha incorporado en nuestro cotidiano. Es el mecanismo por el cual se logra aquello que *existe sólo aparentemente y no es real*.

La virtualización es un término directamente relacionado con la abstracción. Y no es un término ajeno al mundo de los sistemas operativos. En este ámbito, no es un algo nuevo.

La primera versión de un sistema operativo que soportaba máquinas virtuales es de 1972: VM/370 (*Virtual Machine Facility*/370). Pero el concepto ya se venía investigando en la década del 60.

La virtualización es una capa de abstracción sobre el hardware para obtener una mejor utilización de los recursos y flexibilidad. Permite que haya múltiples máquinas virtuales (MV) o entornos virtuales (EV), con distintos (o iguales) sistemas operativos corriendo en forma aislada. Cada MV tiene su propio conjunto de hardware virtual (RAM, CPU, NIC, etc.) sobre el cual se carga el SO huésped (*guest*). Virtualización se transformó en la solución para este mundo de integración que se describió en párrafos anteriores.

Ante situaciones tales como las que se enuncian a continuación, la virtualización se constituye en una solución:

- Existencia de muchas máquinas servidores, poco usados
- Convivencia con aplicaciones heredadas (*legacy*) que no pueden ejecutarse en nuevo hardware o sistema operativo
- Testeo de aplicaciones no seguras
- Ejecución con recursos limitados o específicos
- Necesidad de usar un hardware con el que no se cuenta
- Simulación de redes de computadoras independientes.
- Ejecución de varios y distintos sistemas operativos simultáneamente.

- Testing y monitoreo de performance
- Facilidad de migración.
- Ahorrar energía (tendencias de green IT, o tecnología verde)

Entre otras ventajas, la facilidad de uso se evidencia en el hecho de que las máquinas virtuales están encapsuladas en archivos, lo que las hace fácil de almacenar y de copiar. Sistemas completos (aplicaciones ya configuradas, sistemas operativos, hardware virtual) pueden moverse rápidamente de un servidor a otro.

La dinámica en el caso de los sistemas operativos consiste en un software anfitrión (host) y un software huésped (guest).

Algunas máquinas virtuales en realidad son emuladas. La emulación provee toda la funcionalidad del procesador deseado a través de software. Se puede emular un procesador sobre otro tipo de procesador. Tiende a ser lenta.

La virtualización simula un procesador físico en distintos contextos, donde cada uno de ellos corre sobre el mismo procesador.

En este contexto se hace imprescindible un "manejador/monitor de máquinas virtuales" (VMM, virtual machines monitor) lo que se llama un hipervisor (hypervisor).

En el ámbito de los sistemas operativos, un hipervisor es una plataforma de virtualización que permite múltiples sistemas operativos corriendo en un host al mismo tiempo. El hipervisor interactúa con el HW, realiza la multiprogramación y presenta una o varias máquinas virtuales al sistema operativo.

Hay distintos tipos de hipervisor:

- Hipervisor tipo 1. Se ejecuta en modo kernel. Cada VM se ejecuta como un proceso de usuario en modo usuario. Es decir que hay modo kernel virtual y modo usuario virtual.
- Hipervisor tipo 2. Se ejecuta como un programa de usuario sobre un sistema operativo host. Este host es quien se ejecuta sobre el HW.

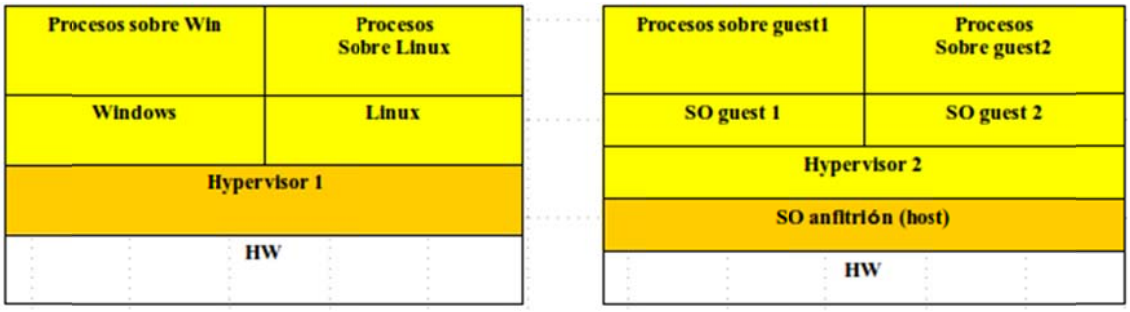

Fig. 29. Organización lógica de un Hipervisor Tipo 1 y uno tipo 2.

A continuación se listan las características de una máquina virtual, enunciadas en 1974 por Popek y Goldberg:

- Equivalencia / Fidelidad: un programa ejecutándose sobre un VMM debería que tener un comportamiento idéntico al que tendría ejecutándose directamente sobre el hardware subyacente.
- Control de recursos / Seguridad: El VMM tiene que controlar completamente y en todo momento el conjunto de recursos virtualizados que proporciona a cada guest.
- Eficiencia / Performance: Una fracción estadísticamente dominante de instrucciones tienen que ser ejecutadas sin la intervención del VMM, o en otras palabras, directamente por el hardware.

Pero analicemos que ocurre a nivel de las instrucciones. Para ello merece hacerse un pequeño repaso.

Una interrupción es un evento que altera la secuencia de instrucciones ejecutada por un procesador. Diferenciamos interrupciones enmascarables de no enmascarables.

La interrupción enmascarable es ignorada por la unidad de control por el tiempo que permanezca en ese estado.

La interrupción no enmascarable será siempre recibida y atendida por la CPU. Sólo unos pocos eventos críticos (como fallos del hardware) pueden generar interrupciones no enmascarables.

Las excepciones son causadas por errores de programación o condiciones anómalas que deben ser manejadas por el *kernel*. Algunas son detectadas por el procesador y otras son programadas. Entre los tipos de excepción están los *traps*.

Recordemos además que generalizando en cuanto a modos de ejecución, en modo supervisor se puede ejecutar cualquiera de las instrucciones que acepta el procesador. En modo usuario, no.

Veamos un ejemplo. En VM/370, hay una máquina CMS (*Conversational Monitor System*) para cada usuario, con “la ilusión” del hardware completo. Una aplicación sobre CMS hace una *system call*, y CMS la “atrapa” (es un *trap*).Y se lo comunica a la MVV.

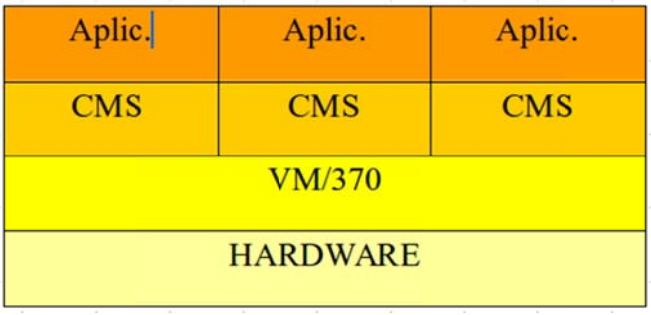

Fig. 30. Organización lógica del IBM VM/370

En la virtualización, el sistema operativo *guest* (ejecutándose en modo usuario) emitirá una instrucción privilegiada que no debe ser ignorada.

INTEL 386 ignoraba las instrucciones privilegiadas que se invocaban en modo usuario. Las máquinas virtuales se ejecutan en modo usuario. Cuando se ejecuta una instrucción privilegiada no debe ignorarse, y debe ser captada por la VMM. Las instrucciones no privilegiadas deben

ejecutarse nativamente (es decir, eficientemente). Popek y Goldberg hicieron un importante aporte al mundo de la virtualización con la definición de instrucciones sensibles e instrucciones privilegiadas. Las instrucciones sensibles siempre generan una excepción y pasan el control al VMM. Las instrucciones privilegiadas generan una interrupción.

Una arquitectura es virtualizable si todas las instrucciones sensibles son privilegiadas.

Cuando se inicia un guest, se ejecuta allí hasta que haga una excepción, que debe ser *atrapada* por el hipervisor.

En el hipervisor tipo 1 cuando la VM ejecuta una instrucción sensible, se produce un trap que procesa el hipervisor.

En los modelos distribuidos que se utilizan hoy en día, la virtualización se ha transformado en un elemento clave, por sus características de aislamiento, fácil administración y ahorro de energía, entre otras ventajas.

3.6. Servicios básicos y extendidos de un sistema operativo distribuido

Cuando pensamos acerca de los servicios básicos y extendidos que provee un sistema operativo, surgen los conceptos de administración de memoria, CPU y hardware como servicios básicos y por ejemplo autenticación y autorización de usuarios como servicios extendidos (entre otros). En entornos distribuidos muchos de los servicios que se consideran básicos para un sistema operativo ya se asumen satisfechos por cada nodo, mientras que surgen otros que resultan de vital importancia con el fin de interoperar con el resto de los nodos que componen el sistema.

En un sistema distribuido, cada nodo posee su propio espacio de memoria, sus dispositivos de entrada salida y sus planificadores. Inclusive, cada nodo representa y administra los datos y recursos, de acuerdo a la arquitectura de hardware y sistema operativo que corra en cada uno. Es de este último concepto, conocido como heterogeneidad que surge una de las premisas más importantes que deben respetar los servicios en entornos distribuidos. Los sistemas deben proveer las correspondientes especificaciones con el fin de poder definir interfaces comunes que permitan construir sobre ellas los servicios que el sistema distribuido brindará. Esto último permitirá lograr la visión de sistema único que interopera con un fin específico.

Dentro de los servicios básicos de un sistema operativo distribuido se pueden considerar:

- **Servicio de comunicación:** “La diferencia más importante entre un sistema distribuido y un sistema de un único procesador es la comunicación entre procesos” (Tanenbaum, 2007). En los sistemas operativos tradicionales, la comunicación entre procesos se realiza a través del pasaje de mensajes asumiendo áreas de memoria compartida. En los sistemas distribuidos, no existe la memoria compartida, ya que cada nodo posee su propia área de memoria que gestiona de modo independiente. La comunicación entre procesos, resulta ahora más compleja y requiere de meca-

nismos que prevean la distribución e inclusive la heterogeneidad. Dichos mecanismos se encuentran plasmados en reglas conocidas como protocolos de comunicación. En líneas generales, cuando un proceso A desea comunicarse con un proceso B residente en otro nodo, el primer proceso construye el mensaje en su propio espacio de direcciones, ejecuta una llamada al sistema para indicarle a su sistema operativo que envíe el mensaje a través de la red y se espera que el proceso B en el nodo remoto reciba los datos. En cada uno de los pasos mencionados, el mensaje atraviesa distintas capas de software que se encargan de *normalizar* el mensaje, de modo tal que pueda ser transmitido y recibido por el destinatario de forma íntegra. Más adelante, en este mismo capítulo, se analizará en detalle el sistema de comunicación y su analogía con las distintas capas de software.

- **Servicio de sincronización:** "Además de la comunicación, es fundamental la forma en que los procesos cooperan y se sincronizan entre sí" (Tanenbaum, 2007). En los sistemas operativos tradicionales, la sincronización es llevada a cabo por el sistema base en un punto centralizado que puede ser consumido por todos los procesos. En los sistemas distribuidos, cada nodo posee su propio reloj, por lo cual surge la necesidad de contar con un mecanismo de sincronización global. Una primera aproximación a la sincronización de relojes en entornos distribuidos, es la que se conoce como *sincronización de relojes físicos*. En esta técnica, el reloj de cada nodo se sincroniza contra un punto central conocido como servidor de hora. De este modo, todos los sistemas operan bajo el mismo esquema horario. Si bien esta aproximación provee buenos resultados, cuando es necesario sincronizar una gran cantidad de nodos, es muy común la producción de desfasajes.

Otra técnica es la utilización de *relojes lógicos*. En esta última, no es necesario contar con una sincronización exacta de los relojes, sino que es más relevante contar con un orden de ocurrencia de los eventos. De este modo, se define la relación *sucedió antes*, que permite determinar que si dos procesos se encuentran relacionados, un evento de uno de ellos suceda antes que otro.

- **Servicio de gestión distribuida de procesos**: La gestión de procesos es una función básica de cualquier sistema operativo. El mismo realiza la gestión en forma centralizada. Sin embargo, cuando se trata de realizar la gestión en un entorno distribuido, existen distintas consideraciones que deben tenerse en cuenta. En un sistema distribuido, los procesadores virtuales se encuentran divididos en los nodos que componen al mismo. Para la ejecución de los procesos, es necesario organizar los procesadores de acuerdo a la arquitectura elegida. Existen tres arquitecturas posibles para realizar la gestión:
    - **Modelo de estaciones de trabajo:** En este esquema, cada proceso se ejecuta en un nodo, mientras que se utilizan sistemas de archivos compartidos para compartir los datos. Este modelo puede provocar que ciertos nodos tengan mucha carga, mientras que otros puedan estar ociosos.

- **Pila de procesadores:** Cada vez que se debe ejecutar un proceso, el mismo es agregado a una pila de donde se seleccionan para su ejecución. De este modo cada nodo que posea recursos para realizar alguna tarea, seleccionará los procesos de la pila global para ser ejecutados.
- **Modelo híbrido:** Los trabajos interactivos son ejecutados bajo el modelo de estaciones de trabajo, mientras que los no interactivos se ejecutan bajo el modelo de pila de procesadores.

En todos los casos es responsabilidad del sistema distribuido realizar una correcta gestión de la carga de trabajo con el fin de optimizar el uso de los recursos.

- **Servicio de memoria compartida distribuida:** En un sistema distribuido, la memoria es gestionada por cada nodo independientemente, pero se comparte lógicamente en el sistema. Cada nodo ejecuta los procesos en su memoria física privada, pero puede consumir recursos de otros nodos que compartan sus espacios de memoria. La forma de acceder a áreas de memoria compartida de otros nodos es a través del pasaje de mensajes. Cada nodo de la red aporta una porción de su memoria local para construir lo que se conoce como el espacio global de direcciones virtuales, el cual será utilizado por los procesos que se ejecuten en el sistema distribuido. El servicio de memoria compartida distribuida se encargará de interceptar las referencias a memoria que realice cada proceso y satisfacerlas ya sea local o remotamente. Los accesos serán enmascarados por el servicio de modo tal de lograr la transparencia y garantizar la heterogeneidad.

  Con el fin de disminuir la sobrecarga en la red por la transferencia de locaciones de memoria, es posible implementar técnicas de caching de las páginas más accedidas. Si bien esta técnica provee mejoras, deben implementarse mecanismos adecuados para evitar incoherencias de los datos. Al momento de realizar actualizaciones de una página, deberá implementarse un protocolo que permita replicar dichos cambios en todas las copias cacheadas localmente por los nodos.

Dentro de los servicios extendidos de un sistema distribuido pueden considerar:

- **Servicio de nombres y localización:** En los sistemas distribuidos, los nombres son utilizados para referirse a recursos como por ejemplo nodos, servicios, usuarios, etc. Los nombres facilitan la comunicación y la compartición de recursos. En líneas generales para que un proceso pueda compartir información con otros, el mismo debe estar nombrado de modo tal que otros puedan localizarlo y accederlo. El ejemplo más común de servicio de nombres es el servicio DNS (Domain Name Service). Se trata de un sistema de nomenclatura jerárquica que permite dar nombre a entidades y realizar búsquedas siguiendo un esquema estructurado.
- **Servicios de Seguridad:** El concepto de seguridad es amplio y abarca a los sistemas distribuidos principalmente en dos aspectos. El primero, referente a la comunicación, tiende a asegurar la comunicación entre procesos residentes en distintos

nodos. La seguridad en la comunicación de procesos que residen localmente en un nodo es llevada a cabo por el sistema operativo. En entornos distribuidos es necesario garantizar la confidencialidad e integridad de los datos transmitidos a través de la red. El segundo aspecto prevé la autorización, la cual debe garantizar que un proceso tenga acceso únicamente a los recursos que le fueron concedidos.

3.7. Conceptos de sistema operativo de red

Cuando nos referimos al término de sistema operativo de red, estamos hablando de una clase especial de software que permite que una red de nodos de computación pueda interoperar entre sí. Utilizando dicho software, los nodos pueden compartir información y recursos.

Dentro de sus funciones principales se encuentran la conexión de los equipos, periféricos y demás dispositivos de red; coordinar las funciones de los nodos y controlar el acceso a los datos y elementos. El sistema operativo de red determina la forma de compartir y acceder a los recursos que gestiona. Se encuentra estrechamente ligado a la arquitectura de red (cliente servidor o trabajo en grupo).

Hoy en día es muy común hablar de sistemas operativos que trabajan en red, y a ellos se los conoce como sistemas operativos de red. Es el caso común de los sistemas operativos Microsoft Windows o GNU/Linux. Netware de Novell, es un sistema operativo de red donde el software de red del equipo cliente se integra al sistema operativo del equipo.

Los sistemas operativos de red se desarrollaron en su gran mayoría siguiendo la arquitectura cliente - servidor. A nivel general, existen dos modelos de arquitectura para su construcción:

- **Modelo de acceso remoto:** Este modelo ofrece a los clientes una forma de acceder transparentemente a los recursos gestionados por algún servidor remoto. Debido a que los clientes desconocen la ubicación de los recursos, el sistema operativo provee mecanismos de localización e interacción con los servicios. La característica más importante de este modelo, es que los datos residen en el servidor remoto y son invocados por los clientes. Un ejemplo de este modelo es el protocolo SSH para acceder remotamente a una línea de comando, o TELNET.
- **Modelo de carga y descarga:** En este caso, el cliente realiza la conexión con el servidor con el fin de descargar los datos del mismo y procesarlos localmente. En la máquina cliente se genera una copia de los datos obtenidos desde el servidor. Una vez que el cliente terminó de utilizar los datos necesarios, vuelve a realizar la carga en el servidor para que las modificaciones se vean reflejadas. Un ejemplo típico de este modelo es el protocolo FTP.

3.8. Relación entre los sistemas distribuidos y las pilas de OSI y TCP/IP

Existen varias definiciones para *Sistema distribuido*, pero en todas se hace referencia a una colección de computadoras conectadas entre sí y apareciendo como un único sistema de cara al usuario. Esta caracterización implica que los sistemas distribuidos presentan ciertas propiedades, como transparencia por ejemplo; más aún, dado que un sistema distribuido se compone de varias partes, cada una realizando una función particular, es posible vislumbrar un paralelismo entre los sistemas distribuidos como un todo y las pilas de red OSI y TCP/IP. Tanto es así que el propósito de esta sección es establecer una analogía entre los sistemas distribuidos en general y los modelos de referencia OSI y TCP/IP.

A modo de repaso, recordemos que el modelo OSI sirve de referencia a la hora de hablar acerca de protocolos de red. Por ejemplo, cuando se dice que IP *es un protocolo de capa 3*, estamos haciendo referencia al modelo OSI y situando a IP en la Capa de Red. Sin embargo, aunque el modelo OSI sea muy conocido y utilizado como referencia, la pila de red más usada actualmente es la denominada TCP/IP (en referencia a los dos protocolos más importantes que la componen).

Tanto TCP/IP como OSI están pensados en términos de *capas* o *layers*, de modo que una capa brinda servicios a la siguiente. Por ejemplo, en TCP/IP, la capa de aplicación usa los servicios brindados por la capa de transporte. Análogamente, los sistemas distribuidos están pensados para que una componente pueda consumir servicios de otras. Esta es otra similitud que sirve para elaborar una analogía entre las pilas de red y los sistemas distribuidos.

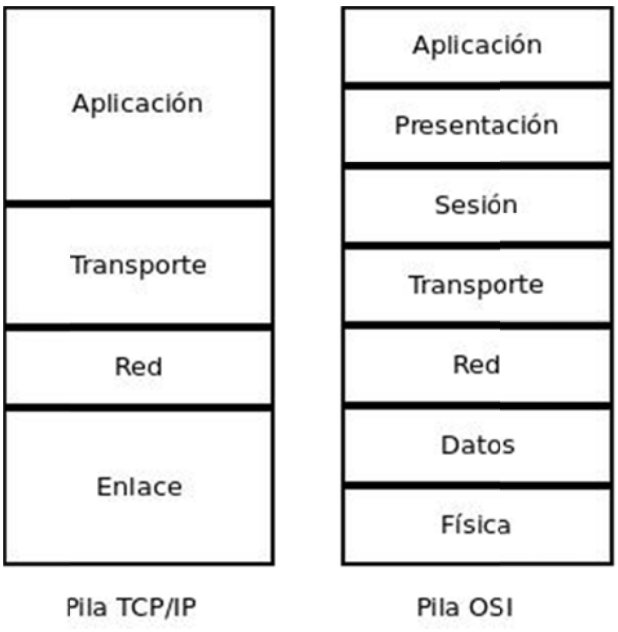

Fig. 31. Capas OSI y TCP

3.9. Sobre las propiedades de un sistema distribuido

Los sistemas distribuidos presentan varias propiedades o características, como son transparencia, apertura (*openness*) y escalabilidad.

Transparencia

Se busca que un sistema distribuido sea transparente para ocultar el hecho de que los recursos y procesos que lo componen se encuentran repartidos entre varias componentes, de modo que los usuarios ven al sistema como un todo. Por ejemplo, cuando un usuario realiza una consulta de DNS, son varias las componentes involucradas en devolver una respuesta (aunque el usuario no percibe esa complejidad).

De forma similar operan las pilas OSI y TCP/IP. En este caso la transparencia se da mediante el encapsulamiento de los datos. De esta manera los datos enviados por una aplicación se rutean hacia su destino, sin que la aplicación conozca las rutas e intermediarios transitados.

Existen varios sentidos en los cuales un sistema puede ser transparente:

- **Acceso:** La transparencia en el acceso implica ocultar la representación de los datos y la manera de accederlos. Por ejemplo, cuando un usuario hace una consulta de DNS, no sabe el formato en el que se almacenan los datos solicitados ni cuáles son los servidores encargados de generar la respuesta, pero de todas maneras recibe una.

  En el caso de los modelos OSI y TCP/IP, las capas inferiores abstraen a las superiores de los pormenores a la hora de acceder a los diferentes medios de comunicación en uso. Por ejemplo, un servidor web que opera en la capa de aplicaciones no sabe si está brindando servicio sobre una red WiFi o cableada, porque las capas inferiores de la pila realizan la abstracción necesaria.
- **Locación:** Se dice que existe transparencia de locación o localidad cuando los recursos puede residir en cualquier lugar sin que esto afecte a la calidad y/o funcionamiento del servicio. En otras palabras, el usuario no sabe dónde está físicamente el recurso en el sistema.

  Algo similar ocurre con los dispositivos conectados a una red. Y esta transparencia de locación la brinda la capa de Red (El protocolo IP en el caso de la pila TCP/IP). Obviamente aquí se asume que la latencia del medio de comunicación es aceptable para el tipo de servicio que se quiere brindar.
- **Concurrencia:** En el contexto de concurrencia transparente se espera que el usuario no perciba el hecho de que un mismo recurso puede estar siendo accedido y usado por otros clientes al mismo tiempo.

  La analogía con las distintas capas de los modelos OSI y TCP/IP es bastante obvia: las distintas capas arbitran el acceso a los recursos y su utilización. Por

ejemplo, la capa de acceso al medio se encarga de multiplexar el tráfico y arbitrar el acceso al medio.

- **Fallas:** Se busca que la aparición y recuperación de fallas sea transparente al usuario. Por ejemplo, si un servidor de DNS falla, automáticamente se pregunta a algún otro servidor que tiene información para la zona en cuestión.

  Análogamente, IP fue pensado para lidiar con fallas en los enlaces, de modo que un paquete puede transitar por rutas alternativas. De todas maneras, IP no asegura la entrega del paquete, es un *protocolo de mejor esfuerzo*. Por otro lado, TCP sí asegura la entrega de todos los datos, o la certeza de que no se pudieron entregar.

Apertura

Por otro lado, cuando se habla de la apertura (*openness* en inglés) de un sistema, se está haciendo referencia al hecho de que el sistema brinda servicios según reglas cuya sintaxis y semántica están bien definidas. A modo de ejemplo se puede tomar la Web, donde existen clientes y servidores web que *hablan* un *idioma* cuya sintaxis y semántica está bien definida: el protocolo HTTP.

Una consecuencia importante de la existencia de estos estándares es la posibilidad de implementarlos sobre diversas plataformas, dando lugar a sistemas distribuidos heterogéneos: No es necesario que todas las componentes corran sobre el mismo hardware, sistema operativo y usen las mismas librerías y lenguajes de programación.

Esta misma noción de servicios bien definidos es lo que permite crear fácilmente nuevos protocolos en las distintas capas de TCP/IP. Por ejemplo, dado que los protocolos UDP y TCP están bien definidos, es relativamente fácil crear protocolos de capa de aplicación que los utilicen.

Escalabilidad

La última propiedad nombrada implica que los sistemas distribuidos deben ser capaces de crecer y contraerse sin perder calidad en el servicio ofrecido: Escalabilidad.

Esta propiedad se puede medir de diferentes maneras. Por un lado, se puede hablar de escalabilidad refiriéndose al tamaño del sistema (cantidad de nodos, datos, usuarios, enlaces). También se habla de escalabilidad geográfica cuando el sistema continúa siendo viable sin importar la lejanía geográfica entre sus usuarios y componentes. Y por último, existe el concepto de escalabilidad administrativa, que implica que las organizaciones se pueden adherir al sistema sin que la calidad del servicio brindado disminuya.

De forma implícita, se espera que la usabilidad y administrabilidad del servicio no disminuya a medida que este crece.

Un ejemplo clásico es el servicio de DNS. La escalabilidad en DNS se da agregando más zonas y servidores al sistema; estos servidores están repartidos geográficamente alrededor del mundo y su administración está delegada a las organizaciones de las zonas correspondientes.

Aquí la analogía puede hacerse con la inclusión de nuevos protocolos a la pila. Por ejemplo, cuando se creó la pila TCP/IP, el protocolo HTTP no existía; sin embargo, fue fácil incluirlo en la capa de aplicaciones.

Otra manera de pensar la escalabilidad en TCP/IP es en la cantidad de nodos que componen el sistema. Las estadísticas[8] muestran el crecimiento de los usuarios de Internet (Y por ende de los usuarios del protocolo IP) a lo largo de los años y está claro que el protocolo fue capaz de soportar ese crecimiento.

3.10. Analogía basada en los servicios

Existen varios modelos que describen la interacción entre las distintas partes de un sistema distribuido. Uno de estos modelos se conoce como Cliente-Servidor, ya descripto en este capítulo. Muchos de los sistemas usados diariamente están implementados según el modelo de Cliente-Servidor. Algunos ejemplos de esta clase de protocolos son HTTP, DNS, SMTP y SSH.

Como se dijo en el párrafo anterior, la modalidad de la interacción consiste en que los clientes envían solicitudes a los servidores, los servidores realizan alguna operatoria para satisfacer la solicitud, y luego envían la respuesta al cliente. De forma similar, cada capa de la pila de red da un servicio a la capa que está inmediatamente encima.

Por definición un sistema distribuido se compone de varias partes y servicios. Es necesario algún mecanismo para referenciar cada una de esas partes. Para esto se utilizan nombres e identificadores.

De forma similar existen nombres e identificadores en las distintas capas de los modelos OSI y TCP/IP. Por ejemplo, la IP de un dispositivo puede verse como un nombre (pero no como un identificador). La MAC sirve al mismo propósito pero en una capa inferior. Así, también, los puertos sirven para referenciar servicios y aplicaciones en un mismo host.

3.11. Conclusiones

En los modelos distribuidos se hace evidente desde lo expuesto anteriormente la idea recurrente de *Divide y vencerás*. Las pilas de red están divididas en capas y los sistemas distribuidos exitosos están divididos en partes.

8 https://en.wikipedia.org/wiki/List_of_countries_by_number_of_Internet_users