Introducción a la computación distribuida de altas prestaciones

Ivan Rodero Castro
Francesc Guim Bernat

PID_00218834

UOC
Universitat Oberta de Catalunya

Índice

Introducción

En este módulo didáctico estudiaremos los conceptos más básicos de sistemas distribuidos y de los dos tipos de sistemas distribuidos que tienen más relación e impacto en la computación de altas prestaciones. Aprenderemos las características de estos sistemas, pero sobre todo, veremos sus diferencias, limitaciones y oportunidades como plataformas para altas prestaciones.

En el primer apartado nos centraremos en los conceptos básicos de los sistemas distribuidos y en las tecnologías básicas utilizadas para implementarlos. En concreto, estudiaremos el RPC, que es un mecanismo de llamada a procedimientos remotos; el RMI, que permite la invocación de métodos remotos de un modo transparente a partir del modelo de objetos distribuidos y, finalmente, los servicios web, que son un mecanismo mucho más abierto y evolucionado y que, entre otras funciones, cumplen un papel importante en el desarrollo de la computación *grid*.

Una vez estudiadas las tecnologías básicas para sistemas distribuidos, nos centraremos en la computación *grid*, que se desarrolla a partir de tecnologías de sistemas distribuidos como las vistas en el primer módulo. En concreto, estudiaremos las características de los sistemas de computación *grid* y posteriormente utilizaremos un ejemplo de *middleware* y otro de metaplanificación (*meta-scheduling*) para ilustrar las funcionalidades más significativas de estos entornos.

Finalmente, estudiaremos la computación *cloud*. Del mismo modo que veremos que la motivación de la computación *grid* es la colaboración entre instituciones para desarrollar avances científicos a nivel internacional, también veremos que el desarrollo de la computación *cloud* está basada en criterios económicos y utiliza conceptos de mercado. Estudiaremos tanto las características principales de los sistemas *cloud* –la elasticidad y el modelo de pago de recursos por uso, por ejemplo–, como los diferentes tipos de *clouds*, los diferentes modelos de servicio y los conceptos fundamentales de virtualización, utilizando Xen como caso de estudio. Concluiremos analizando los posibles usos y retos abiertos para utilizar la computación *cloud* para computación de altas prestaciones.

Objetivos

Los principales objetivos que alcanzaréis con el estudio de este módulo son los siguientes:

1. Conocer los fundamentos de la computación distribuida y saber diferenciarla de la computación de altas prestaciones tradicional.

2. Aprender las tecnologías básicas de sistemas distribuidos, como RPC, RMI y servicios web, que posibilitan el desarrollo de sistemas distribuidos a escala, como la computación *grid* o *cloud*.

3. Entender la motivación del desarrollo de la computación *grid* y saber identificar los componentes esenciales, como el concepto de organización virtual, el *middleware* y el *meta-scheduling*.

4. Entender las diferencias primordiales entre los sistemas de computación *grid* y los sistemas de altas prestaciones tradicionales.

5. Entender la motivación del desarrollo de la computación *cloud* y saber identificar los componentes esenciales, como el concepto de elasticidad, pagar por uso o la virtualización.

6. Aprender los tipos de *clouds* más importantes y los posibles modelos de servicio.

7. Entender las diferencias entre los sistemas de computación *cloud* y *grid* y los sistemas de altas prestaciones tradicionales, y cuáles son algunas de las posibilidades para aprovechar/combinar la computación *cloud* con la de altas prestaciones.

1. Fundamentos de la computación distribuida

En este apartado estudiaremos los fundamentos más básicos de la computación distribuida y nos centraremos en las tecnologías más relevantes, que son la base y posibilitan el desarrollo de la computación *grid* y *cloud*, y que, a pesar de no seguir el modelo clásico de computación de altas prestaciones, son los sistemas distribuidos que tienen un papel significativo en la computación de altas prestaciones. Os animamos a hacer una búsqueda sobre otros sistemas distribuidos, como el sistema *peer-to-peer* o modelos de computación que también se pueden aplicar a sistemas distribuidos para computación de altas prestaciones, como el *autonomic computing*.

1.1. Introducción a la computación distribuida

Un **sistema distribuido** se define como un conjunto de computadores autónomos conectados por una red y con el software distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar capacidad de computación.

El desarrollo de los sistemas distribuidos llegó de la mano de las redes locales de alta velocidad a principios de los setenta. Más recientemente, la disponibilidad de ordenadores personales de altas prestaciones, estaciones de trabajo y ordenadores servidores ha derivado en un mayor desplazamiento hacia los sistemas distribuidos en detrimento de los ordenadores centralizados multiusuario. Esta tendencia se ha acelerado por el desarrollo de software para sistemas distribuidos, diseñado para soportar el desarrollo de aplicaciones distribuidas. Este software permite a los ordenadores coordinar sus actividades y compartir los recursos del sistema, es decir, el hardware, el software y los datos.

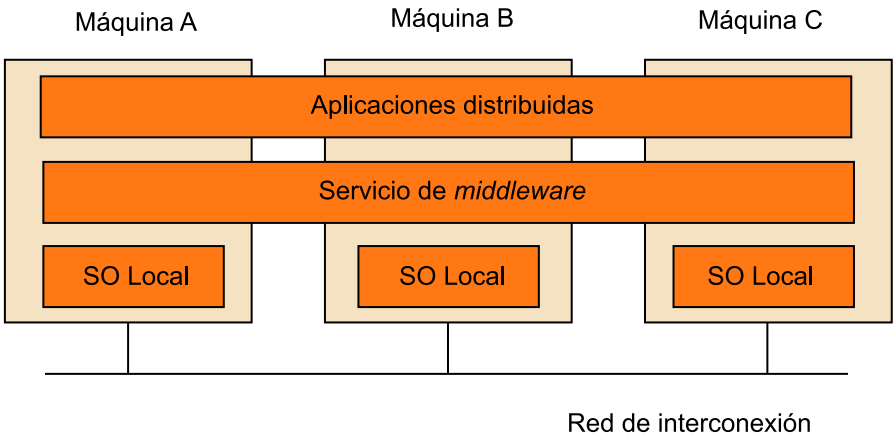
Los sistemas distribuidos se implementan en varias plataformas hardware, desde unas pocas estaciones de trabajo conectadas por una red de área local, hasta Internet, que no deja de ser un conjunto de redes de área local y de gran alcance interconectadas que enlazan millones de ordenadores.

Las aplicaciones de los sistemas distribuidos varían desde la provisión de capacidad de cómputo a grupos de usuarios, hasta sistemas bancarios, comunicaciones multimedia e incluso prácticamente todas las aplicaciones comerciales y técnicas de los ordenadores. Los requisitos de las aplicaciones incluyen un alto nivel de fiabilidad, seguridad contra interferencias externas y privacidad de los datos. Se deben proveer accesos concurrentes a bases de datos por parte de muchos usuarios, garantizar tiempos de respuesta, proveer puntos de acce-

so al servicio que están geográficamente dispersos, etc., lo que provoca que los sistemas distribuidos tengan un potencial muy elevado para las empresas para crecer y realizar negocio.

El objetivo de un sistema distribuido es integrar los recursos y servicios conectados por una red de interconexión, tal y como se puede ver en la figura 1. Desde el punto de vista del usuario y de las aplicaciones, un sistema distribuido proporciona una visión de máquina única y no se diferencia de uno centralizado. En cambio, desde el punto de vista del diseñador (el sistema como gestor de los recursos) la estructura interna está condicionada por la distribución física de los recursos.

Figura 1. Estructura de un sistema distribuido genérico



Lo más habitual es que el sistema operativo integre los servicios de red que ofrecen protocolos abiertos de comunicación, como TCP y UDP. Sobre estos se disponen los soportes adicionales para la comunicación distribuida, como es el caso de RPC, RMI o DSM[1], y los servicios específicos que proporcionan las propiedades del sistema distribuido (servicios de *middleware*), como es el caso de la gestión de tiempo, eventos y estado global, sobre los que se fundamentan las aplicaciones.

[(1)]En inglés, *distributed shared memory*.

Actividad

Buscad información sobre DSM y pensad las similitudes y las limitaciones respecto a otros modelos de memoria distribuida.

Para definir los sistemas distribuidos se establecen seis características principales:

- Compartición de recursos.
- Apertura.
- Concurrencia.
- Escalabilidad.
- Tolerancia a fallos.
- Transparencia.

1.1.1. Compartición de recursos

El término *recurso* es bastante abstracto, pero es el que mejor caracteriza el conjunto de entidades que pueden compartir en un sistema distribuido. Este conjunto incluye desde componentes hardware como discos e impresoras, hasta elementos software como ficheros, bases de datos y otros objetos de datos.

La idea de compartición de recursos no es nueva ni aparece en el marco de los sistemas distribuidos. Los sistemas multiusuario clásicos desde siempre han proveído compartición de recursos entre sus usuarios. Los recursos de un ordenador multiusuario se comparten de manera natural entre todos sus usuarios. Por el contrario, los usuarios de estaciones de trabajo monousuario u ordenadores personales dentro de un sistema distribuido no obtienen automáticamente los beneficios de la compartición de recursos.

Los recursos en un sistema distribuido están físicamente encapsulados en uno de los computadores y solo se puede acceder a ellos por otros computadores mediante comunicación vía red. Para que la compartición de recursos sea efectiva, esta debe ser gestionada por un programa que ofrezca una interfaz de comunicación que permita que se pueda acceder al recurso y que sea manipulado y actualizado de una manera fiable y consistente. Aquí surge el concepto de gestor de recursos.

Un **gestor de recursos** es un módulo software que gestiona un conjunto de recursos de un tipo en particular. Cada tipo de recurso requiere algunas políticas y métodos específicos junto a requisitos comunes para todos ellos. Estos incluyen la provisión de un esquema de nombres para cada clase de recurso, permitir que se pueda acceder a los recursos individuales desde cualquier localización, la traducción del nombre de recurso a direcciones de comunicación y la coordinación de los accesos concurrentes que cambian el estado de los recursos compartidos para mantener la consistencia.

Un sistema distribuido se puede ver de manera abstracta como un conjunto de gestores de recursos y un conjunto de programas que usan estos recursos. Los usuarios de los recursos se comunican con los gestores de los recursos para acceder a los recursos compartidos del sistema. Esta perspectiva nos conduce a dos modelos de sistemas distribuidos: el modelo cliente-servidor y el modelo basado en objetos.

1.1.2. Apertura

Un sistema informático es **abierto** si el sistema puede ser extendido de varias maneras. Un sistema puede ser abierto o cerrado en cuanto a extensiones hardware (añadir periféricos, memoria, interfaces de comunicación, etc.) o respecto a las extensiones software (añadir características al sistema operativo, protocolos de comunicación y servicios de compartición de recursos, etc.).

La apertura de los sistemas distribuidos se determina en primer lugar por el grado en el que se pueden añadir nuevos servicios de compartición de recursos sin perjudicar ni duplicar los ya existentes. Algunas características de los sistemas distribuidos que hay que tener en cuenta respecto a la apertura son:

- Las interfaces software clave del sistema están claramente especificadas y se ponen a disposición de los desarrolladores, es decir, las interfaces se hacen públicas.

- Los sistemas distribuidos abiertos se basan en la provisión de un mecanismo uniforme de comunicación entre procesos e interfaces publicadas para acceder a recursos compartidos.

- Los sistemas distribuidos abiertos pueden construirse a partir de hardware y software heterogéneo, posiblemente proveniente de vendedores diferentes. Pero la conformidad de cada componente con el estándar publicado debe ser cuidadosamente comprobada y certificada si se quiere evitar tener problemas de integración.

1.1.3. Concurrencia

Cuando hay varios procesos en una única máquina decimos que se están **ejecutando concurrentemente**. Si el computador está equipado con un único procesador central, la concurrencia tiene lugar entrelazando la ejecución de los diferentes procesos. Si el computador tiene N procesadores, entonces se pueden estar ejecutando estrictamente a la vez hasta N procesos.

En los sistemas distribuidos hay muchas máquinas, cada una con uno o más procesadores centrales. Es decir, si hay M ordenadores en un sistema distribuido con un procesador central cada una, entonces hasta M procesos pueden estar ejecutándose en paralelo.

En un sistema distribuido que está basado en el modelo de compartición de recursos, la posibilidad de ejecución paralela pasa por dos razones:

1) Muchos usuarios interactúan simultáneamente con programas de aplicación.

2) Muchos procesos servidores se ejecutan concurrentemente, cada uno respondiendo a diferentes peticiones de procesos clientes.

El primer caso es menos conflictivo, dado que normalmente las aplicaciones de interacción se ejecutan aisladamente en la estación de trabajo del usuario y no entran en conflicto con las aplicaciones ejecutadas en las estaciones de trabajo de otros usuarios.

El segundo caso surge debido a la existencia de uno o más procesos servidores para cada tipo de recurso. Estos procesos se ejecutan en diferentes máquinas, de manera que se ejecutan en paralelo varios servidores, junto con varios programas de aplicación. Las peticiones para acceder a los recursos de un servidor especificado pueden formar cola en el servidor y ser procesadas secuencialmente, o pueden ser procesadas concurrentemente por múltiples instancias del proceso gestor de recursos. Cuando esto sucede, los procesos servidores deben sincronizar las acciones para asegurarse de que no se producen conflictos. La sincronización debe ser planeada cuidadosamente para asegurar que no se pierden los beneficios de la concurrencia.

1.1.4. Escalabilidad

Los sistemas distribuidos operan de manera efectiva y eficiente en muchas escalas diferentes. La escala más pequeña consiste, por ejemplo, en dos estaciones de trabajo y un servidor de ficheros, mientras que un sistema distribuido construido sobre una red de área local simple podría contener varios centenares de estaciones de trabajo, varios servidores de ficheros, servidores de impresión y otros servidores de propósito específico. A menudo se conectan varias redes de área local para formar otras mayores, y estas podrían contener muchos miles de ordenadores que forman un único sistema distribuido, permitiendo que los recursos sean compartidos entre todos ellos.

Tanto el software de sistema como el de aplicación no deberían cambiar cuando la escala del sistema aumenta. La necesidad de escalabilidad no es solo un problema de prestaciones de red o de hardware, sino que está íntimamente ligada a todos los aspectos del diseño de los sistemas distribuidos. El diseño del sistema debe hacerse partiendo de la idea de que será necesario que escale; en caso contrario, habrá muchas limitaciones.

La demanda de escalabilidad en los sistemas distribuidos ha conducido a una filosofía de diseño en la que cualquier recurso, ya sea hardware o software, se puede extender para proporcionar servicio a tantos usuarios como se quiera.

Esto significa que si la demanda de un recurso crece, debería ser posible extender el sistema para dar servicio. Por ejemplo, la frecuencia con la que se accede a los ficheros aumenta cuando se incrementa el número de usuarios y estaciones de trabajo en un sistema distribuido. Entonces, ha de ser posible añadir servidores para evitar el cuello de botella que se produciría si un solo servidor de ficheros tuviera que gestionar todas las peticiones de acceso a los ficheros. En este caso, el sistema debe estar diseñado de manera que permita trabajar con ficheros replicados en diferentes servidores, con las consideraciones de consistencias que esto supone.

Cuando el tamaño y la complejidad de las redes crece, es un objetivo primordial diseñar software distribuido, que seguirá siendo eficiente y útil con estas nuevas configuraciones de la red.

Resumiendo, el trabajo necesario en el proceso de una petición para acceder a un recurso compartido debería ser prácticamente independiente del tamaño de la red. Las técnicas necesarias para conseguir estos objetivos incluyen el uso de datos replicados, técnicas relacionadas con memorias caché y el uso de múltiples servidores para gestionar las tareas que permitan aprovechar la concurrencia y, por lo tanto, obtener más productividad.

1.1.5. Tolerancia a fallos

Los sistemas informáticos a veces fallan. Cuando se producen errores en el software o en el hardware, los programas podrían producir resultados incorrectos o podrían interrumpirse antes de terminar su computación.

El diseño de **sistemas tolerantes a fallos** se basa en dos cuestiones complementarias entre sí: redundancia de hardware (uso de componentes redundantes) y recuperación del software (diseño de programas que sean capaces de recuperarse de los errores).

En los sistemas distribuidos, la redundancia puede plantearse en un grano más fino que el hardware, se pueden replicar los servidores individuales que son esenciales para la operación continuada de aplicaciones críticas. La recuperación del software tiene relación con el diseño de software que sea capaz de recuperar el estado de los datos permanentes antes de que se produzca el fallo.

La disponibilidad en los sistemas distribuidos también tiene gran importancia y muchas veces su falta está relacionada con fallos de hardware. La disponibilidad de un sistema es una medida de la proporción de tiempo que está disponible para su uso. Un simple fallo en una máquina multiusuario provoca la no disponibilidad del sistema para todos los usuarios. Cuando uno de los

componentes de un sistema distribuidos falla, solo se ve afectado el trabajo que estaba realizando el componente averiado. Un usuario podría desplazarse a otra estación de trabajo y un proceso servidor podría ejecutarse en otro computador.

1.1.6. Transparencia

La **transparencia** se define como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un todo, en vez de un conjunto de componentes independientes.

La transparencia ejerce una gran influencia en el diseño del software de sistema. Podemos hablar de ocho tipos de transparencia:

1) Transparencia de acceso: permite el acceso a los objetos de información remotos del mismo modo que a los objetos de información locales.

2) Transparencia de localización: permite el acceso a los objetos de información sin conocimiento de su localización.

3) Transparencia de concurrencia: permite que varios procesos operen concurrentemente, utilizando objetos de información compartidos y de modo que no haya interferencia entre ellos.

4) Transparencia de replicación: permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación deban conocer la existencia de las réplicas.

5) Transparencia de fallos: permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.

6) Transparencia de migración: permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.

7) Transparencia de prestaciones: permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varía.

8) Transparencia de escalado: permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

Las dos más importantes son las transparencias de acceso y de localización; su presencia o ausencia afecta gravemente a la utilización de los recursos distribuidos. A menudo se las denomina **transparencias de red**.

1.2. Modelo cliente-servidor

El modelo cliente-servidor de un sistema distribuido es el modelo más conocido y más ampliamente adoptado en la actualidad. Hay un conjunto de procesos del servidor, cada uno actuando como un gestor de recursos para un conjunto de recursos de un tipo, y un conjunto de procesos cliente, cada uno llevando a cabo una tarea que requiere acceso a algunos recursos hardware y software compartidos.

Los gestores de recursos podrían necesitar acceder a recursos compartidos gestionados por otros procesos, así que algunos procesos pueden ser tanto clientes como servidores. En el modelo cliente-servidor, todos los recursos compartidos son mantenidos y gestionados por los procesos servidor. Los procesos cliente realizan peticiones a los servidores cuando necesitan acceder a algún recurso. Si la petición es válida, entonces el servidor lleva a cabo la acción requerida y envía una respuesta al proceso cliente.

El modelo cliente-servidor nos da un enfoque efectivo y de propósito general para la compartición de información y de recursos en los sistemas distribuidos. El modelo puede ser implementado en una gran variedad de entornos software y hardware. Los computadores que ejecutan los programas cliente y servidor pueden ser de muchos tipos y no hay la necesidad de distinguirlos, dado que los procesos cliente y servidor pueden incluso estar en la misma máquina.

En esta aproximación del modelo cliente-servidor, cada proceso servidor podría ser visto como un proveedor centralizado de los recursos que gestiona. No obstante, la provisión de recursos centralizada no es deseable en los sistemas distribuidos. Por esa razón se distingue entre los servicios proporcionados a los clientes y los servidores encargados de proveer estos servicios. Se considera un servicio como una entidad abstracta que puede ser provista por varios procesos servidores que funcionan en varios computadores y cooperan a través de la red.

El modelo cliente-servidor se ha extendido y utilizado en los sistemas actuales con servicios que gestionan numerosos tipos diferentes de recursos compartidos, como correo electrónico y mensajes de noticias, ficheros, sincronización de relojes, almacenamiento en disco, impresoras e incluso las interfaces gráficas de usuario. Pero no es posible que todos los recursos que existen en un sistema distribuido sean gestionados y compartidos de este modo; algunos tipos de recursos deben permanecer locales en cada computador de cara a una mejor eficiencia, como en el caso de la memoria RAM, un procesador, una

interfaz de red local, etc. Estos recursos clave son gestionados separadamente por un sistema operativo en cada computador; solo podrían ser compartidos entre procesos localizados en el mismo computador.

1.2.1. *Middleware*

El software distribuido requerido para facilitar las interacciones cliente-servidor se denomina ***middleware***. El acceso transparente a servicios y recursos distribuidos mediante una red de interconexión se realiza mediante el *middleware*, que sirve como marco para las comunicaciones entre los componentes cliente y servidor de un sistema.

El *middleware* define la interfaz que usan los clientes para pedir un servicio a un servidor, la transmisión física de la petición mediante la red y la devolución de resultados desde el servidor al cliente. Algunos ejemplos de *middleware* estándar para dominios específicos son: ODBC para bases de datos, HTTP y SSL para Internet, y CORBA y RMI para objetos distribuidos.

El *middleware* fundamental o genérico es la base de los sistemas cliente-servidor. Los servicios de autenticación en red, llamadas a procedimientos remotos, sistemas de ficheros distribuidos y servicios de tiempos en red se consideran parte del *middleware* genérico. El *middleware* específico para un dominio complementa el *middleware* genérico para aplicaciones mucho más específicas. El protocolo de comunicaciones más utilizado por el *middlware*, tanto genérico como específico, es TCP/IP debido a su gran difusión.

1.3. Llamada a procedimiento remoto (RPC)

El paso de mensajes permite expresar el modelo cliente-servidor explicitando un protocolo de petición-respuesta sobre el servicio al que se quiere acceder. En cambio, el acceso a recursos en los sistemas operativos tradicionales mediante la interfaz de llamadas al sistema se basa en una semántica de llamada-retorno a funciones. Siendo la transparencia en la ubicación de los recursos un objetivo fundamental en los sistemas distribuidos, resulta evidente que el paso de mensajes no es una base semántica adecuada para el acceso a los recursos remotos.

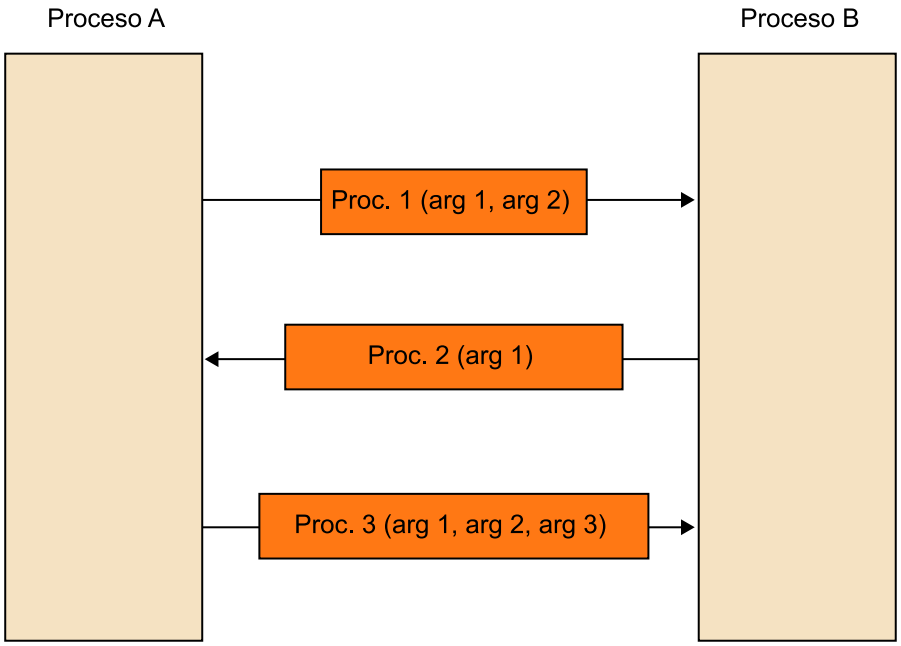
El mecanismo de llamada a procedimiento remoto (RPC)[2] trata de eliminar este salto semántico. Un proceso cliente especifica el acceso a un servicio mediante una sintaxis de llamada a función, por ejemplo:

[2] Del inglés *remote procedure call.*

```
resultado = servicio_rpc (parámetros)
```

Un servicio proporcionado por un servidor no es más que un conjunto de operaciones disponibles para los clientes. El acceso al servicio se realiza mediante un protocolo de peticiones y respuestas con llamadas bloqueantes. Un ejemplo de ello es el sistema de ficheros, en el que el servidor mantiene como recurso compartido los ficheros y sobre el que se pueden realizar varias operaciones, como crear, abrir, leer, etc. La figura 2 muestra un ejemplo genérico de llamadas RPC.

Figura 2. Ejemplos de llamadas a procedimientos remotos



Los mecanismos **RPC** se encargan de proporcionar a los clientes una capa de abstracción para llamar a procedimientos remotos (operaciones) y así obtener servicios.

De este modo, el procedimiento especificado se ejecuta en otro proceso de otra máquina (servidor). El objetivo de RPC es mantener la semántica de la llamada a procedimientos normales en un entorno de implementación totalmente diferente. La ventaja está en que el desarrollador solo debe preocuparse de las interfaces que soporta el servidor. Para especificar estas interfaces se dispone de un lenguaje de definición de interfaces (IDL)[3].

[(3)]Del inglés *interface description language*.

Los sistemas RPC disponen de mecanismos de RPC integrados en un lenguaje de programación particular que incluye además una notación para definir interfaces entre clientes y servidores (IDL específico). Un IDL permite definir el nombre de las operaciones soportadas por el servidor y sus parámetros (tipos y dirección). También se deben proveer mecanismos para el manejo de excepciones y garantizar la ejecución de las operaciones y la detección de fallos. Todo esto del modo más transparente posible.

El software *(middleware)* que soporta RPC tiene tres tareas fundamentales:

1) Procesamiento relacionado con las interfaces: integrar RPC en el entorno de programación, empaquetamiento[4]/desempaquetamiento[5] y atender las peticiones al procedimiento adecuado.

2) Gestionar las comunicaciones.

3) Enlazar[6]: localizar al servidor de un servicio.

(4)En inglés, *marshalling*.

(5)En inglés, *unmarshalling*.

(6)En inglés, *binding*.

La interfaz no es más que un nombre de procedimiento acordado entre cliente y servidor. Este nombre viaja dentro del mensaje RPC transmitido por la red. En la parte del cliente hay un procedimiento de *stub* encargado de empaquetar/desempaquetar los argumentos de la llamada y convertir la llamada local en otra remota. Esto supone mandar un mensaje, esperar la respuesta y devolver los resultados. En la parte del servidor está el expedidor, junto al conjunto de procedimientos de *stub* de servidor, que tienen una misión similar a los de la parte del cliente. El expedidor selecciona el procedimiento de *stub* adecuado a partir del nombre de procedimiento requerido.

El compilador de IDL genera los procedimientos de *stub* de cliente y de servidor, las operaciones de empaquetamiento/desempaquetamiento y los ficheros de cabecera necesarios. Las peticiones de los clientes se hacen con un nombre de servicio. En última instancia deben ser dirigidas a un puerto en el servidor. En un sistema distribuido, un *binder* es un servicio que mantiene una tabla que contiene correspondencias de nombres de servicios con puertos de servidor. Se trata de hecho de un servicio de nombres. Importar un servicio es pedir al binder que busque el nombre de la interfaz y devuelva el puerto del servidor. Exportar un servicio es registrar de cara el *binder*. El *binder* deberá estar en un puerto muy conocido o, cuando menos, se lo podrá localizar.

Una implementación bastante habitual de RPC es la de SUN. Esta incorpora un conjunto de primitivas para trabajar con RPC en lenguaje C. Dispone de una representación neutral de los datos (XDR)[7] para su empaquetamiento.

(7)Del inglés *external data representatio*.

El compilador de IDL, que se denomina *rpcgen*, genera los procedimientos de *stub*, el procedimiento *main* del servidor y el expedidor, el código de conversión de los parámetros en XDR y los ficheros de cabecera correspondientes.

El procedimiento completo para la ejecución de una RPC es el siguiente:

1) A partir del nombre del servicio remoto, el cliente localiza el servidor que soporta el servicio.

2) Se empaquetan los argumentos en un formato estándar (por ejemplo, XDR20) para formar el cuerpo de un mensaje, es decir, se serializan[8].

(8)En inglés, *marshaling*.

3) Se utiliza la interfaz de red del sistema operativo para mandar un mensaje al servidor, que contiene la petición del servicio (por ejemplo, mediante UDP/IP).

4) El proceso cliente queda bloqueado en la recepción de la respuesta.

5) En el nodo destino, al recibir el mensaje, el sistema operativo desbloquea al servidor. Si este es multihilo, un hilo de ejecución se hará cargo de la petición.

6) Se ejecuta el mecanismo que desempaqueta el mensaje (deserialización) para obtener los parámetros de la petición de servicio y la identificación del origen.

7) Se ejecuta la función asociada al servicio solicitado, que habrá sido instalada por el servidor en su inicialización.

8) La función del servidor se ejecuta mediante llamadas al sistema local.

9) Cuando acaba la función asociada al servicio, se serializa el resultado.

10) Se envía el mensaje con el resultado al cliente.

11) El sistema operativo del nodo del cliente desbloquea a este en la recepción del resultado.

12) El cliente ejecuta la deserialización del resultado.

13) Se devuelve el valor en el programa.

1.4. Invocación de métodos remotos (RMI)

La invocación de métodos remotos (RMI)[9] corresponde al modelo de programación orientada a objetos, al contrario de las RPC, que pertenecen al modelo de programación procedimental. Primero veremos los conceptos relacionados con la invocación de métodos del modelo de objetos, centrándonos después en el modelo de objetos distribuidos.

(9)Del inglés *remote method invocation*.

1.4.1. Modelo de objetos

En los lenguajes de programación orientada a objetos, como C++ y Java, un objeto, que encapsula un conjunto de datos y de métodos, se comunica con otros objetos invocando los métodos de estos objetos, que en general aceptan argumentos y devuelven resultados.

Aunque estos lenguajes proporcionan modos para permitir referenciar directamente las variables de los objetos, en el modelo de objetos distribuidos, que veremos después, las variables solo pueden acceder mediante métodos.

Los objetos se acceden por referencia. Al invocar un método de objeto (que denominaremos *objeto receptor*) hay que proporcionar la referencia al receptor, el método y los argumentos de la invocación. Las referencias a objetos se pueden asignar a variables, pasar como argumentos o devolver como resultados de una invocación. Una interfaz define el formato de acceso a un conjunto de métodos, es decir, sus nombres, tipos de argumentos, valores de regreso y excepciones (pero no la implementación de los métodos).

En la invocación de un método, el objeto receptor ejecuta el método y retorna el control, devolviendo eventualmente un resultado. Como consecuencia, el estado del receptor puede cambiar. También se pueden desencadenar invocaciones a otros objetos. Durante la ejecución del método se pueden producir condiciones de error (excepciones).

1.4.2. Modelo de objetos distribuidos

En un sistema distribuido, los objetos de las aplicaciones pueden estar repartidos entre los nodos del sistema. Como sucede en las RPC, la invocación de métodos remotos normalmente se realiza mediante el modelo cliente-servidor, donde los objetos receptores se gestionan por los servidores de estos objetos. La invocación mantiene la transparencia en la ubicación del objeto.

En el **modelo de objetos distribuidos**, los procesos constan de objetos que se comunican mediante invocaciones locales o remotas. Las locales son invocaciones a métodos del mismo objeto, mientras que las invocaciones remotas son a métodos de objetos de otros procesos, ya estén en el mismo nodo o en otros nodos del sistema.

Algunos objetos solo pueden recibir invocaciones locales, mientras que otros, los objetos remotos, pueden recibir tanto invocaciones locales como remotas. Para invocar un método de un objeto remoto, un objeto debe tener acceso a la referencia de objeto remoto del receptor, que es un identificador único en el sistema distribuido

Cada objeto remoto tiene una interfaz remota que especifica cuáles de sus métodos se pueden invocar remotamente. Estos métodos estarán implementados por la clase del objeto remoto. En Java RMI, las interfaces remotas se definen como toda interfaz de Java, sin más que ampliar la interfaz “Remote”. CORBA proporciona un lenguaje de definición de interfaces (CORBA IDL), que permite que las clases de los objetos remotos y los programas clientes estén progra-

mados en lenguajes diferentes. En una invocación remota se pueden producir, además de las generales, excepciones relacionadas con la naturaleza remota de la invocación, como por ejemplo *timeouts*.

1.4.3. Java RMI

Java RMI amplía el modelo de objetos de Java para soportar objetos distribuidos de un modo integrado en el lenguaje, haciendo la invocación de métodos remotos transparente, excepto por el hecho de que el cliente deba tratar las excepciones remotas y el servidor definir como "Remote" la interfaz del objeto remoto.

Las interfaces remotas se definen ampliando la interfaz "Remote", que está proporcionada por el paquete "java.rmi". Los parámetros de un método son de entrada, y la salida se proporciona en el resultado de la invocación. Los objetos remotos se pasan por referencia y los locales por valor, mediante serialización. Cuando el receptor no dispone de la implementación del objeto que se le pasa por valor, la máquina virtual Java proporciona la descarga automática de la clase correspondiente.

Los nodos que contienen objetos remotos proporcionan un servicio de nombres que almacena las referencias de objetos remotos, el registro de objetos remotos ("rmiregistry").

En una aplicación distribuida en Java RMI, hay que definir las interfaces remotas e implementar los objetos remotos y los clientes. Una vez compilados los ficheros fuente, la aplicación se organiza de la siguiente manera:

1) Se crean los *proxies* (*stubs*, en terminología Java-RMI) de las clases remotas mediante el compilador de RMI ("rmic").

2) Se especifica el acceso a las clases y se establece una política de seguridad.

3) Se pone en marcha el registro de objetos remotos como un proceso del servidor ("rmiregistry").

4) Se pone en marcha el servicio remoto y se lanzan los clientes.

Implementación de un método que suma dos números en Java RMI

A continuación se muestra un ejemplo de implementación de un método que suma dos números en Java RMI.

Primero se define la interfaz que declara el método remoto, que tiene dos números como argumentos y devuelve la suma.

```
import java.rmi.*;

public interface AddServerIntf extends Remote {
   double add(double d1, double d2) throws RemoteException;
}
```

Después se implementa el código correspondiente a la interfaz remota y el servidor.

```
import java.rmi.*;
import java.rmi.server.*;

public class AddServerImpl extends UnicastRemoteObject
   implements AddServerIntf {

   public AddServerImpl() throws RemoteException {
   }
   public double add(double d1, double d2) throws RemoteException {
  return d1 + d2;
   }
}
```

El siguiente código se encarga de hacer que los objetos estén disponibles para nodos remotos, es decir, que actualiza el registro RMI (mediante "rebind").

```
import java.net.*;
import java.rmi.*;

public class AddServer {
   public static void main(String args[]) {
     try {
        AddServerImpl addServerImpl = new AddServerImpl();
        Naming.rebind("AddServer", addServerImpl);
     }
     catch(Exception e) {
        System.out.println("Exception: " + e);
     }
   }
}
```

Finalmente, hay que implementar un programa cliente que será el encargado de utilizar la interfaz remota y acabar llamando al método del objeto remoto. Este cliente tiene tres argumentos: la dirección IP, o el nombre del servidor remoto, y los dos números que se han de sumar.

```
import java.rmi.*;
public class AddClient {
   public static void main(String args[]) {
     try {
        String addServerURL = "rmi://" + args[0] + "/AddServer";
        AddServerIntf addServerIntf =
          (AddServerIntf)Naming.lookup(addServerURL);
        System.out.println("The first number is: " + args[1]);
        double d1 = Double.valueOf(args[1]).doubleValue();
        System.out.println("The second number is: " + args[2]);

        double d2 = Double.valueOf(args[2]).doubleValue();
     System.out.println("The sum is: " + addServerIntf.add(d1, d2));
     }
    catch(Exception e) {
        System.out.println("Exception: " + e);
     }
  }
}
```

1.5. Servicios web

El concepto de servicio web[10] ha evolucionado desde su definición inicial. Los inicios de los servicios web los podemos encontrar en los sistemas distribuidos, con CORBA y DCOM, pero ninguno de los dos sistemas acabó generalizándose debido a los problemas de aplicación en la Internet real. Más adelante, la W3C predefiniría los servicios web con el protocolo SOAP[11], que forma parte

(10) En inglés, *web service.*

(11) Del inglés *simple access protocol.*

(12) En inglés, *grid computing.*

de la capa de mensajería. Desde su creación se ha extendido rápidamente su funcionamiento y ha sido fundamental para tejer tanto la Web 2.0 como otras infraestructuras, como la computación *grid*[12].

Actualmente, los servicios SOAP destinados a aplicaciones distribuidas conviven con nuevas formas más simples y no estándares debido al mejor rendimiento en la implementación de SOAP. XML-RPC, JSON-RPC y PHP-RPC son los nuevos métodos/protocolos/formatos más destacados que encontramos en los servicios web en las interfaces de, por ejemplo, Facebook, Google y Yahoo.

Entre las distintas organizaciones que se encargan de velar por los estándares y arquitectura web destacan el WWWC (W3C[13]) y OASIS[14].

(13)De World Wide Web Consortium.

(14)De The Organization for the Advancement of Structured Information Standards.

El W3C define los servicios web del siguiente modo:

"Un servicio web es un sistema de software diseñado para dar soporte a la interacción interoperable de máquina a máquina en una red. Tiene una interfaz descrita en un formato procesable por máquina, específicamente WSDL. Otros sistemas interactúan con el servicio web de una manera prescrita por su descripción, usando mensajes SOAP, típicamente transmitido a través de HTTP con una serialización XML en conjunción con otras normas relacionadas con la web".

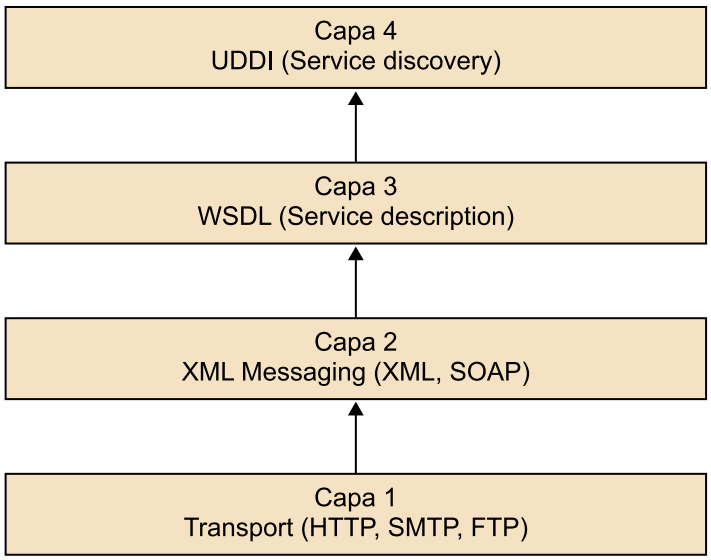
El W3C se puede definir más claramente como un modo estandarizado de integrar aplicaciones web mediante el uso de XML, SOAP, WSDL y UDDI. Estos permiten la comunicación entre aplicaciones o componentes de estas aplicaciones de manera estándar, mediante protocolos comunes (típicamente HTTP) y de manera totalmente independiente al lenguaje de programación, plataforma de implantación, sistema operativo o formato de presentación. El éxito de la interoperabilidad se logra con la adopción de protocolos y estándares abiertos.

En un intento de estandarización global, el W3C propone su visión de los servicios web, especificando la *web services protocol stack* (figura 3), que es un conjunto de protocolos para servicios web. A pesar del peso y la importancia del W3C como organismo de estandarización en los servicios web, entre otras tecnologías, es cierto que muchos fabricantes proponen pequeños cambios a esta especificación.

Web services protocol stack

Es un conjunto de protocolos y estándares para redes (Internet, intranet, etc.) utilizados para definir, localizar, implementar y hacer que un servicio web interactúe con otros.

Figura 3. *Web services protocol stack*



La capa de transporte es responsable del transporte de los mensajes entre las aplicaciones de red y los protocolos, como HTTP, HTTPS, SMTP, FTP, JABBER, IIOP, BEEP, etc. En la práctica, la gran mayoría de los servicios web están disponibles para HTTP y/o HTTPS.

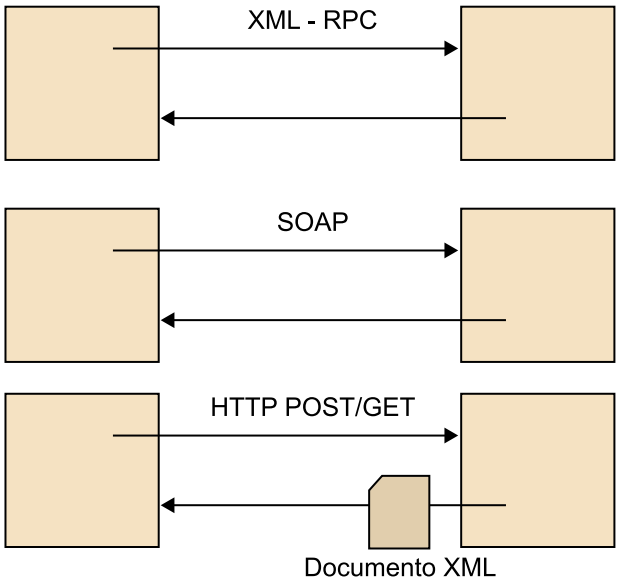
La **mensajería XML** consiste en un lenguaje de marcas ampliable. Es un estándar para describir datos y crear etiquetas. Las características especiales son la independencia de datos y la separación de los contenidos de su representación. Es un metalenguaje que permite diseñar un lenguaje propio de etiquetas para múltiples tipos de documentos. Los documentos XML están formados por unidades de almacenamiento, conocidas como entidades, que contienen datos analizados[15] o sin analizar[16]. Los datos analizados están formados por caracteres. Algunos de estos caracteres forman los datos del documento y el resto forma las etiquetas.

(15)En inglés, *parsed*.

(16)En inglés, *unparsed*.

Las etiquetas codifican la descripción de la estructura lógica y de almacenamiento del documento. XML proporciona un mecanismo para imponer restricciones a la estructura lógica y de almacenamiento. XML se convirtió en el estándar de las comunicaciones para Internet. Lo que antes se transmitía en formatos propietarios ahora es fácilmente interoperable gracias a XML. En la especificación de la W3C solo se contempla XML y SOAP como capa de mensajería (figura 4).

Figura 4. Mensajería XML



El **WSDL**[17] es un lenguaje de descripción de servicios web. Es una especificación XML para la construcción del documento de descripción del servicio web. Cuando decimos que es una especificación XML significa que el documento WSDL está escrito en XML.

(17)Del inglés *web service description language.*

El fichero WSDL identifica los métodos, las funciones y los parámetros necesarios para invocar un determinado servicio y describe información crítica que el cliente del servicio web necesita conocer, como:

- El nombre del servicio, incluyendo su URN[18].
- La localización del servidor, normalmente una dirección URL[19] por HTTP.
- Los métodos disponibles para ser invocados.
- Los parámetros de entrada y salida para cada uno de los métodos.

(18)Del inglés *uniform resource name*.

(19)Del inglés *uniform resource location*.

Ejemplo de fichero WSDL

A continuación se muestra un ejemplo de fichero WSDL, que describe el servicio del típico "Hello World" en el que se define el método "sayHello".

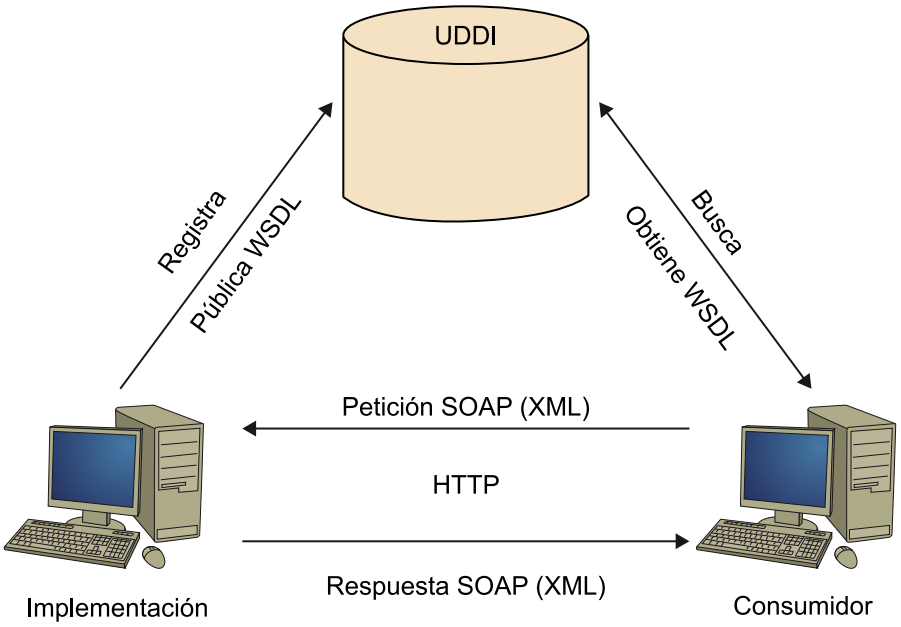
```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
```

```
        <operation name="sayHello">
           <input message="tns:SayHelloRequest"/>
           <output message="tns:SayHelloResponse"/>
        </operation>
     </portType>

     <binding name="Hello_Binding" type="tns:Hello_PortType">
     <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
     <operation name="sayHello">
        <soap:operation soapAction="sayHello"/>
        <input>
           <soap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:examples:helloservice"
              use="encoded"/>
        </input>
        <output>
           <soap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:examples:helloservice"
              use="encoded"/>
        </output>
     </operation>
     </binding>

     <service name="Hello_Service">
        <documentation>WSDL File for HelloService</documentation>
        <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address
           location="http://www.examples.com/SayHello/">
        </port>
     </service>
  </definitions>
```

UDDI[20] es un elemento fundamental en el que se apoyan los servicios web. Este estándar ofrece la posibilidad de que los proveedores/clientes de servicios puedan publicar y encontrar otros servicios web de terceros. UDDI tiene un mecanismo para que los proveedores de servicios se describan a sí mismos y los diferentes servicios que proporcionan, permitiendo que se puedan registrar y publicar en un registro UDDI. Estos servicios publicados pueden ser buscados, consultados o descubiertos por otros usando mensajes SOAP. La figura 5 muestra el funcionamiento de UDDI.

(20)Del inglés *universal description discovery and integration*.

Figura 5. Funcionamiento de UDDI



Los datos tratados por UDDI se clasifican en tres categorías:

- **Páginas blancas**: contienen información general de la empresa/organización, como nombre, descripción, información de contacto, dirección y teléfono.

- **Páginas amarillas**: es muy parecido a su equivalente telefónico; incluyen categorías de catalogación industrial, ubicación geográfica, etc. Existen unos códigos y claves preestablecidas que facilitan la inscripción en el registro y así se facilita a terceros la búsqueda de servicios mediante estos códigos de clasificación.

- **Páginas verdes**: contienen información técnica sobre un servicio web. Normalmente incluye un puntero a la especificación externa y una dirección en la que invocar el servicio.

SOAP es un protocolo de acceso simple a objetos. Es una especificación XML para la creación de los mensajes intercambiados entre sistemas distribuidos y la red. Este protocolo se deriva inicialmente del XML-RPC. Los mensajes están formados por un sobre (ENVELOPE), la cabecera (HEADER) y el cuerpo (BODY). El sobre envuelve el mensaje y contiene la cabecera y el cuerpo. La cabecera es opcional y solo da información para el enrutamiento del mensaje.

Ejemplo

En los documentos XML siguientes se muestran una petición y una respuesta SOAP, respectivamente.

Petición SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
   <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
   </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

Respuesta SOAP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
   <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
   </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

2. Computación *grid*

En este apartado se introducen los conceptos más fundamentales de la computación *grid*[21]. Utilizaremos algunas soluciones concretas para ejemplificar las características de las funcionalidades de la computación *grid*. Por último, discutiremos los aspectos fundamentales que diferencian los sistemas clásicos de altas prestaciones de la computación *grid*. Se espera que complementéis los contenidos de este apartado con investigación propia a partir de información y lecturas que se os proporcionarán durante el curso.

(21) En inglés, *grid computing*.

2.1. Introducción a la computación *grid*

En las últimas décadas y gracias especialmente a los avances en las tecnologías de la información, la ciencia está cambiando el modo como se lleva a cabo. Anteriormente, la contribución científica podía producirse a partir de un grupo relativamente pequeño de investigadores que trabajaban conjuntamente en el laboratorio. En cambio, los retos científicos actuales requieren que múltiples grupos de investigación colaboren a escala internacional y compartan todo tipo de información y recursos.

De este modo, la idea esencial de la computación *grid* es la de compartir recursos entre diferentes grupos de investigación y, en general, entre diferentes instituciones u organizaciones. En la computación *grid*, los recursos están interconectados mediante redes de interconexión de gran alcance y utilizan una capa de software que permite el acceso transparente a los recursos. De hecho, el término *grid* proviene del símil entre la idea de compartir recursos distribuidos de manera transparente con la red eléctrica[22], donde los usuarios consumen energía de la red sin saber su fuente o cómo se ha generado (por ejemplo, si proviene de fuentes renovables o no). Este modelo consiste en proporcionar un servicio como utilidad[23].

(22) En inglés, *power grid*.

(23) En inglés, *utility computing*.

Podemos definir la **computación *grid*** como un modo de computación distribuida, que ofrece la abstracción de un único computador a partir de un grupo heterogéneo de recursos (computadores, instrumentos, almacenamiento, etc.) interconectados a través de redes de gran alcance (Internet).

Un sistema de computación *grid* está compuesto por múltiples equipos informáticos que trabajan juntos, sobre una infraestructura de hardware y software, compartiendo recursos disponibles, pero a la vez independientes entre sí. Esto significa que cada recurso se gestiona de manera autónoma, respetando las políticas locales (por ejemplo, las políticas de planificación de trabajos).

A pesar de las distancias físicas a las que se pueden encontrar las diferentes máquinas que componen el *grid*, el poder de unificación y el modo de relación dinámica de estas máquinas consiguen responder a todas las demandas de los diferentes usuarios de la red. Dicho esto, se puede ver que las peticiones de los usuarios se distribuyen en diferentes tareas para diferentes equipos que forman el *grid*, de modo que el usuario simplemente ve los datos finales como un solo cómputo.

Hay que tener en cuenta que el origen de la computación *grid* está íntimamente ligado a la compartición de recursos entre la comunidad científica, que requiere altas prestaciones para resolver problemas complejos. Un ejemplo claro de ello es el desarrollo de la computación *grid* durante la última década para poder dar soporte al LHC[24] en el CERN de Ginebra. Parte del reto técnico-científico es poder almacenar y procesar los datos generados en un tiempo razonable y lograr realizar descubrimientos científicos de dominios concretos, como la física de partículas.

[24]Del inglés *large hadron collider*.

Una de las grandes ventajas de la computación *grid* es que se puede montar sobre la infraestructura de red y de recursos ya existente, lo que permite reducir enormemente el gasto inicial. Los recursos de la computación *grid* son extremadamente heterogéneos, es decir, tanto a nivel de hardware como de software podemos encontrar una gran diversidad de plataformas diferentes: diferentes arquitecturas de procesadores, sistemas operativos, aplicaciones, etc. Esto tiene importantes repercusiones a la hora de definir el software encargado de gestionar el *grid*; en primer lugar, la planificación de recursos (decidir sobre qué recursos se ejecuta una determinada tarea) resulta un problema bastante complejo. Por otro lado, es muy importante la utilización de estándares de comunicación y de interfaces que permitan una buena compatibilidad entre software tan diferente.

Los recursos del *grid* no son dedicados, dado que son recursos ya existentes en la institución y posiblemente se usen para uso personal o institucional (por ejemplo, un grupo de investigación de la universidad) a la hora de formar parte del *grid*. Así pues, la cantidad de recursos que están disponibles para el *grid* varía en función de la utilización local. Esto hace imprescindible incorporar una monitorización constante a los recursos y políticas de replanificación con capacidad para variar el conjunto de recursos asignados a una tarea en tiempo de ejecución.

La dispersión geográfica y la gran cantidad de recursos que participan en el *grid* provoca que el número de usuarios potenciales sea muy elevado y con cambios constantes. Además, puede haber múltiples instituciones involucradas en el *grid*, lo que se gestiona a partir del concepto de organización virtual, tal y como veremos a continuación.

2.2. Concepto de organización virtual

Foster (2001) introdujo el concepto de **organización virtual** (**VO**). Define la organización virtual como "una colección dinámica de múltiples organizaciones, que proporcionan flexibilidad y el intercambio seguro y coordinado de recursos".

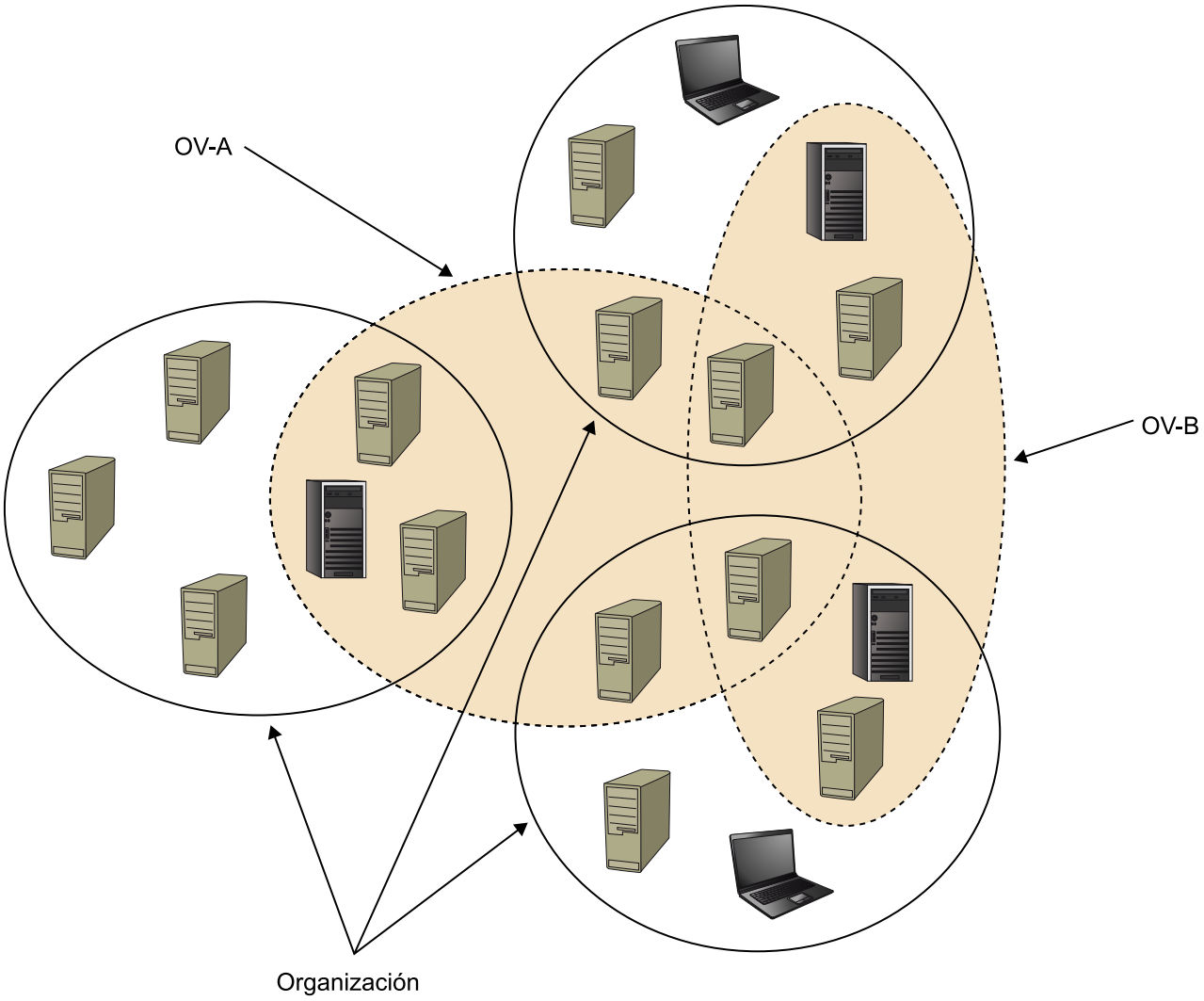
Lectura complementaria

I. Foster; C. Kesselman; S. Tuecke (2001). "The Anatomy of the Grid, Enabling Scalable Virtual Organizations". *International Journal of High Performance Computing Applications* (vol. 15, núm. 3).

La figura 6 muestra tres organizaciones reales con recursos tanto computacionales como de datos para compartir a través de las fronteras organizacionales. Por otro lado, la figura forma dos organizaciones virtuales, A y B, y cada una de ellas puede tener acceso a un subconjunto de los recursos en cada una de las organizaciones. La virtualización es un mecanismo que mejora la utilidad de los sistemas de computación *grid*, proporcionando a los usuarios la posibilidad de configurar su entorno.

Hay que tener en cuenta que para poder implementar múltiples organizaciones virtuales por encima de organizaciones físicas, hay que utilizar mecanismos de seguridad y gestión de usuarios, tal y como veremos más adelante. Aun así, el concepto de organización virtual coincide claramente con el espíritu colaborativo con el que se inició el desarrollo de la computación *grid*.

Figura 6. Ejemplo de dos organizaciones virtuales



2.3. El *middleware*

De toda la gestión y control de la red se encarga el *middleware*, el software instalado en los recursos del *grid* que está entre el sistema operativo y las aplicaciones que utilizan el *grid*. Hay una serie de cuestiones comunes, como los esquemas de seguridad (autenticación de usuarios y recursos), que suelen implementarse en todos los *middleware*, pero normalmente hay recursos en la red destinados a la gestión del *grid*, los denominados *brokers*.

El *middleware* de los recursos de un *grid* también tiene por objetivo realizar otras tareas, como la replicación de los datos y los metadatos, que requerirán que los nodos ejecuten una cierta tarea por cuestiones de eficiencia o tolerancia a fallos. Para simplificar este tipo de tareas se definieron una serie de protocolos especializados en redes para *grid*, como el GridFTP, que veremos con detalle más adelante.

En este subapartado nos centraremos en Globus Toolkit como ejemplo de *middleware*, dado que se convirtió en el estándar *de facto*. Aun así, hay que tener en cuenta que existen otros *middlewares* para la computación *grid*, como Unicore, Condor o gLite.

Actividad

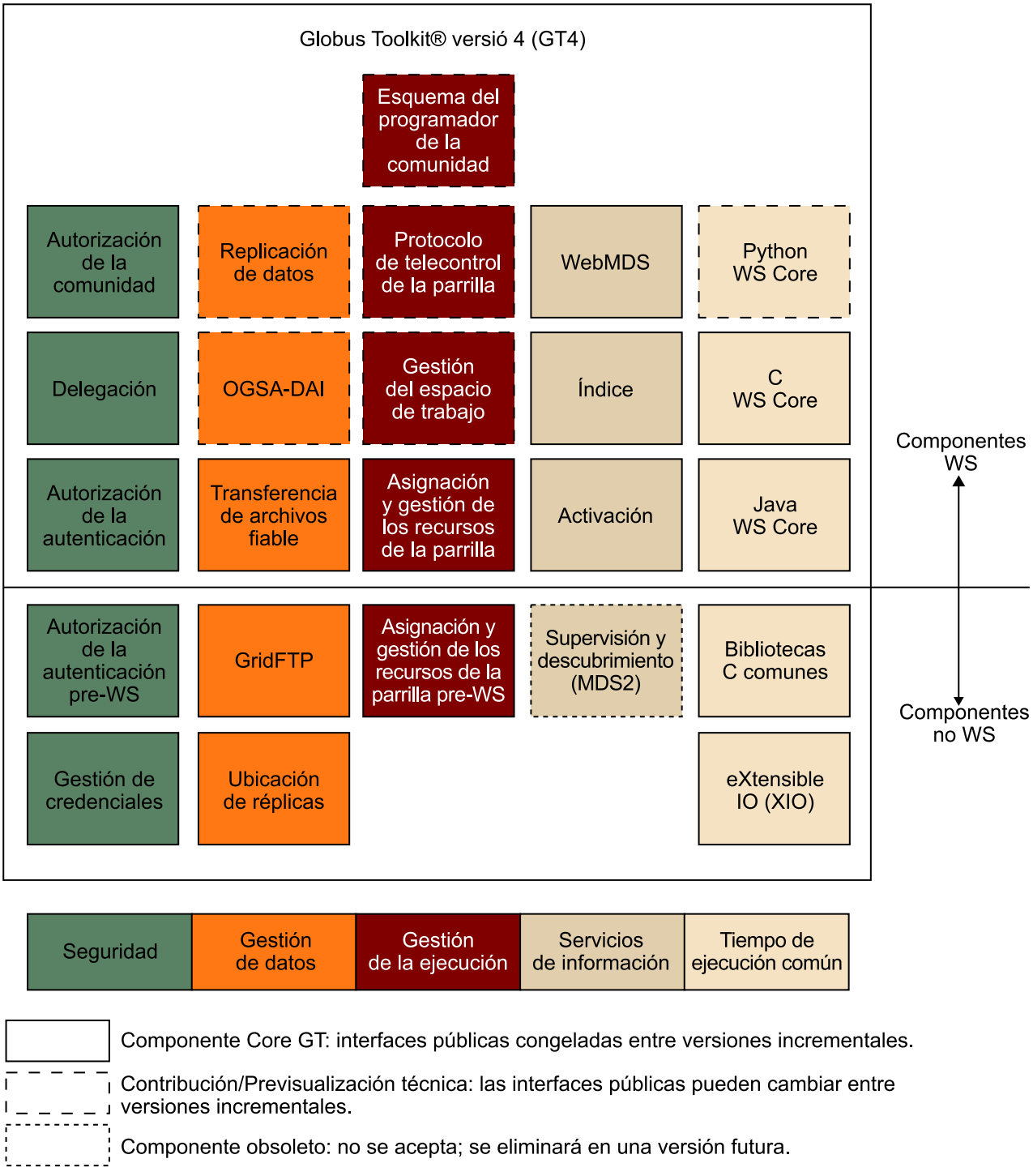
Buscad información adicional sobre diferentes *middleware* para computación *grid* y analizad sus pros y contras.

La Globus Alliance es el nombre de un esfuerzo internacional entre diferentes instituciones e individuos cuyos objetivos son la investigación y el desarrollo de tecnologías y estándares para la computación *grid*. Una de sus tareas principales es la construcción del Globus Toolkit, un conjunto de soluciones para problemas típicos de la computación *grid*, como mantener la seguridad en entornos distribuidos y heterogéneos, gestión de recursos, monitorización, solución de fallos y gestión de datos, entre otros, que permiten construir *middleware* para cubrir las necesidades de gestión del *grid*. El desarrollo del Globus Toolkit ha tenido desde el principio un enfoque de software libre y construcción descentralizada, así como una clara preferencia para la adopción de estándares. También hay que destacar el uso de servicios web a nivel de interfaz entre módulos, basada en mecanismos XML, consiguiendo un sistema distribuido poco acoplado[25].

(25)En inglés, *loosely coupled*.

La arquitectura del Globus Toolkit se puede diferenciar en cinco pilares básicos, como podemos ver en la figura 7. Por un lado, los cuatro fundamentos de la gestión del *grid* en forma de bibliotecas, incluyendo seguridad, gestión de datos, gestión de la ejecución y servicios de información. Además de esto, el Globus Toolkit ofrece interfaces para diferentes lenguajes de programación (C, Java y Python) que permiten desarrollar servicios web basados en el resto de los servicios ofrecidos por el Globus Toolkit. Un último grupo de componentes ofrecidos por el Globus Toolkit son las implementaciones de servicios (aquellos por debajo de la línea negra en la figura, etiquetados como "Componentes no WS"). Como su nombre indica, son implementaciones de soluciones propias del *grid* (por ejemplo, GridFTP, GRAM, etc.) sin estar basadas en servicios web.

Figura 7. Diagrama de los módulos que forman el Globus Toolkit



Componente Core GT: interfaces públicas congeladas entre versiones incrementales.

Contribución/Previsualización técnica: las interfaces públicas pueden cambiar entre versiones incrementales.

Componente obsoleto: no se acepta; se eliminará en una versión futura.

2.3.1. Gestión de la ejecución

El principal servicio ofrecido por el Globus Toolkit en materia de gestión de la ejecución es el servicio de gestión y reparto de recursos del *grid* (GRAM)[26], que proporciona una interfaz para enviar, monitorizar y cancelar tareas. De este modo, terceras partes pueden basarse en estas implementaciones, por ejemplo para construir planificadores de tareas.

(26)Del inglés *grid resource allocation and management*.

Lectura complementaria

Podéis encontrar mucha más información sobre la arquitectura interna de este módulo en el documento "Globus Toolkit 4.0 WS GRAM Approach".

2.3.2. Gestión de los datos

Dentro del mundo de la computación *grid*, la transferencia de datos es un problema fundamental, pues nos encontramos con conjuntos de nodos computacionales repartidos por localizaciones geográficas muy dispersas. Además, ciertos tipos de aplicaciones necesitan un uso intensivo de datos, como es el caso del LHC construido en el CERN.

El Globus Toolkit ofrece distintas piezas de software que dan solución a algunos de los problemas relacionados con la transferencia y el acceso a grandes cantidades de datos. Por un lado, la implementación de la especificación GridFTP proporciona herramientas para hacer transferencias seguras y de alto rendimiento de memoria a memoria o disco a disco, y además tiene compatibilidad con clientes y servidores FTP tradicionales. El servicio de replicación de ubicaciones[27] proporciona un servicio descentralizado para mantener información sobre la ubicación de conjuntos de datos y ficheros repartidos por la red. Otra solución, como el servicio de transferencia fiable de ficheros[28], proporciona una nueva capa sobre el GridFTP que permite hacer transferencias múltiples y fiables de ficheros a gran escala.

(27)En inglés, *replica location service*.

(28)En inglés, *reliable file transfer*.

2.3.3. Servicios de información

En cuanto a monitorización y descubrimiento de los recursos en *grid*, Globus Toolkit ofrece herramientas para convertir los datos relacionados sobre los recursos en entidades XML que describen las características de estos. Estos servicios, basados en las especificaciones WSFR y WS-Notification, se complementan con otras de indexación, que no están basados en los anteriores estándares (para aquellos nodos que utilicen un sistema donde estos no estén implementados).

2.3.4. Seguridad

Las propiedades de los entornos de computación *grid*, principalmente su dispersión geográfica de usuarios y recursos de diferentes instituciones, organismos, entidades, etc., hacen de la seguridad un tema importante. Establecer la identidad de usuarios y recursos, definir y aplicar políticas de seguridad o reforzar la seguridad de las comunicaciones son los pilares básicos de una política de seguridad de la computación *grid*. En el Globus Toolkit encontraremos

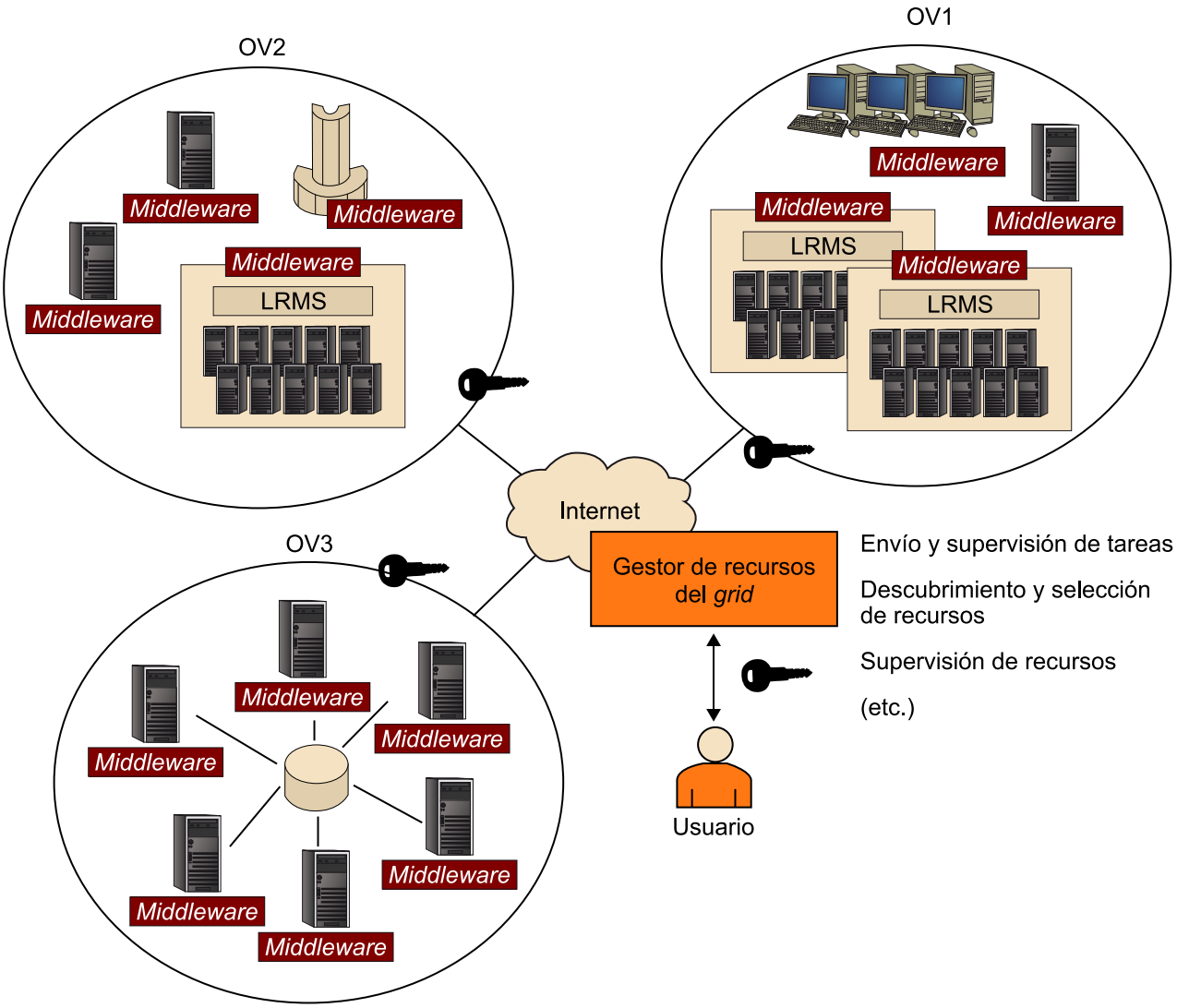
componentes de seguridad de bajo nivel que implementan protocolos para la protección de mensajes, autenticación, delegación y autorización. Como se muestra en la figura 7, se proporciona apoyo para soluciones que cumplen con el estándar WS-Security con credenciales X-509, así como con usuarios y contraseñas, y para seguridad a nivel de transporte con credenciales X.509 En la configuración por defecto de Globus Toolkit 4, cada usuario y cada recurso tiene una clave pública X.509 que permite a los diferentes protocolos realizar la validación entre dos entidades, así como establecer un canal seguro de comunicación entre ellas.

2.4. *Meta-scheduling*

En la computación *grid*, las tareas son aplicaciones preparadas para su ejecución, empaquetadas junto con los datos de entrada necesarios y la descripción de sus necesidades tecnológicas. Si el *middleware* sigue los estándares, utilizamos el *job submission description language*, una especificación de XML que nos permite definir aspectos como los requisitos que deben tener los computadores para poder ejecutarla (memoria RAM disponible, *swap* disponible, CPU, sistema operativo, etc.), variables de entorno necesarias para su correcta ejecución, así como los datos de los que depende, etc.

Una de las funciones más importantes de la gestión del *grid* consiste en decidir cuáles serán los recursos que utilizará una determinada tarea. Dada la diversidad de recursos que forman el *grid*, así como su heterogeneidad (diferentes sistemas operativos, arquitecturas, software, etc.) y su dinamismo, este es un trabajo que requiere unos algoritmos de planificación más propios de un sistema operativo distribuido. Esta es la función del *meta-scheduler* o *broker*, una parte del *middleware* que se ocupa de monitorizar los nodos y de planificar la ejecución de las tareas sobre ellos de la manera más efectiva posible. Habitualmente también cuenta con métodos de recuperación de fallos, y para evitar los puntos únicos de fallo, en muchos de los casos se encuentran distribuidos entre más de un nodo de la red, o trabajando de un modo redundante. En el caso de la figura 8, el *meta-scheduler* (*grid resource broker*) es centralizado.

Figura 8. Arquitectura *grid* con un *meta-scheduler* centralizado



Aunque hay docenas de *meta-schedulers* para sistemas de computación *grid*, en este subapartado veremos los aspectos más generales de GridWay, que es uno de los *meta-schedulers* más conocidos.

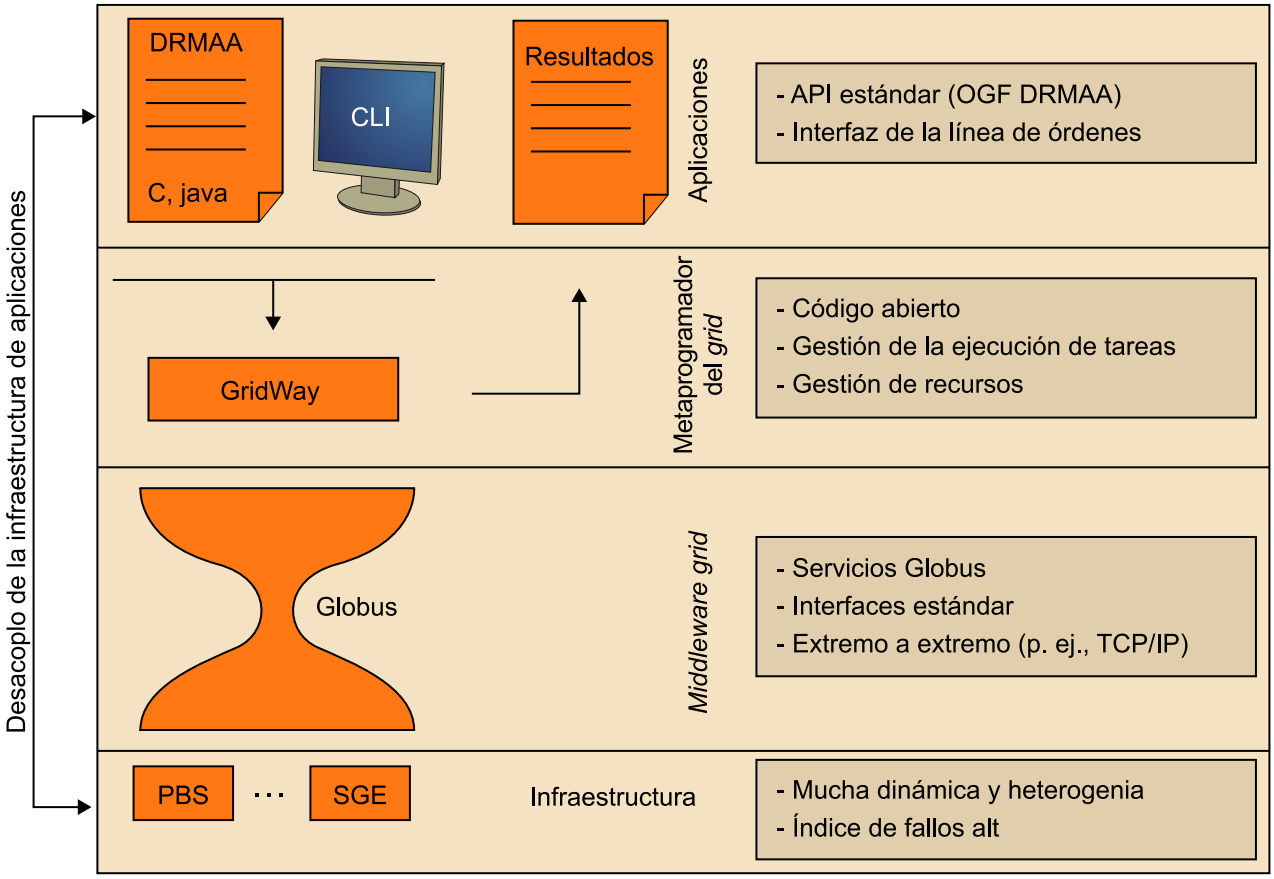
Actividad

Buscad información adicional sobre diferentes *meta-schedulers* para computación *grid* y estudiad sus características. También podéis buscar artículos a modo de resumen sobre *meta-schedulers* en *grid*.

GridWay es una solución desarrollada en la Universidad Complutense de Madrid. Este *meta-scheduler*, licenciado como software libre y creado bajo la filosofía de la Globus Alliance, está pensado para funcionar con servicios de Globus, como el Globus Toolkit.

La figura 9 muestra el lugar que ocupa el *meta-scheduler* GridWay en la infraestructura del *grid*, virtualizando un servicio de planificación y consiguiendo separar la infraestructura de las aplicaciones que pueden ejecutarse sobre el *grid*.

Figura 9. Capas que forman una red en *grid* utilizando GridWay



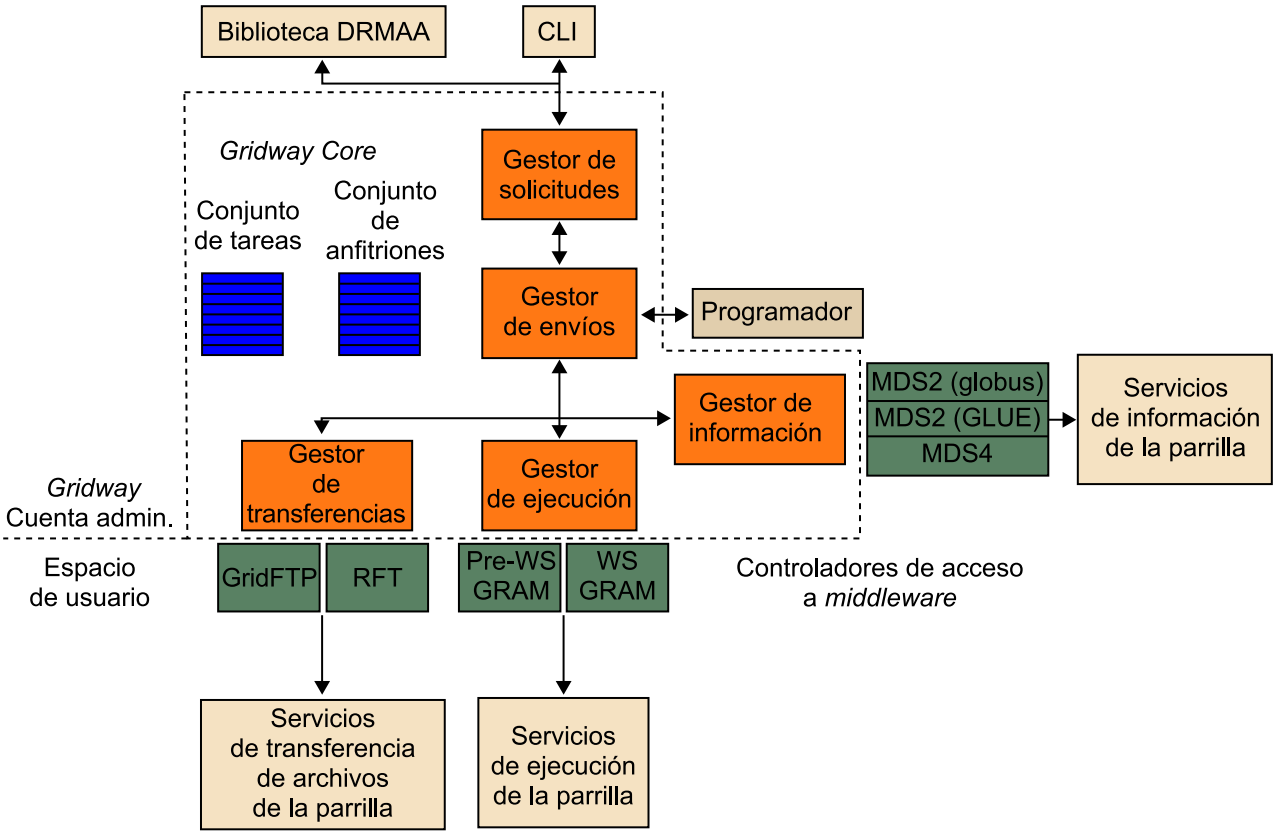
Tal y como se muestra en la figura 10, GridWay proporciona una interfaz de usuario basada en línea de pedidos, pero también permite utilizar la especificación Distributed Resource Management Application (DRMAA), establecida por el Open Grid Forum y cuyo objetivo es que las propias aplicaciones puedan interactuar con el *grid* directamente.

GridWay nos ofrece una gran cantidad de políticas de planificación, algunas de las cuales describiremos brevemente a continuación. Veremos estas políticas a modo de ejemplo, aunque existen muchas políticas de varios tipos con diferentes objetivos de optimización, tanto para GridWay como para otros *meta-schedulers*. La tarea de la política de planificación es, como su nombre indica, decidir el orden de ejecución de las tareas en el Grid.

Actividad

Buscad información adicional sobre políticas de planificación para *meta-schedulers* para computación *grid*. También podéis buscar artículos a modo de resumen sobre políticas de *meta-scheduling* en el *grid*.

Figura 10. Diagrama de la arquitectura de GridWay



A continuación enumeramos tres políticas basadas en prioridades estáticas en las tareas de un usuario o de un grupo en concreto, o *fixedpriority policy* (*FP*). El rango va desde 00 a 19 y el envío de tareas al *grid* prioriza las de valor inferior:

1) La política *fair-share policy* permite establecer unas ratios de prioridad para el envío de tareas al *grid* para los diferentes usuarios. Por ejemplo, podemos establecer una ratio 10:4 para las tareas de un grupo de investigación o departamento determinado. El problema de esta política es que no tiene en cuenta el uso de los recursos, un aspecto fundamental a la hora de aprovechar de manera adecuada un sistema de computación *grid*.

2) La política *waiting-time policy* tiene por objetivo evitar que las tareas con una menor prioridad nunca pasen a ejecutarse para que otras de mayor prioridad monopolicen los recursos continuamente (problema de asignación de recursos compartidos conocido en el mundo de la computación como inanición). Lo que hace es incrementar linealmente la prioridad de las tareas a medida que pasa el tiempo mientras espera a ser ejecutada.

3) Los límites temporales de ejecución[29] se pueden marcar gracias a la política *deadline policy*, que incrementa la prioridad de ejecución de una tarea a medida que se acerca. Aun así, es importante señalar que no estamos hablando de un planificador de tiempo real, pues esta política no asegura que las tareas se ejecuten antes del tiempo especificado.

(29)En inglés, *deadlines*.

Aparte de estas políticas basadas en prioridad, merece la pena destacar el mecanismo denominado *matchmaking*, que es utilizado en muchas políticas y se desarrolló en el contexto de Condor. En pocas palabras, *matchmaking* se encarga de realizar un emparejamiento de los trabajos con los recursos de manera que se satisfagan sus requisitos.

Actividad

Buscad información sobre técnicas de *matchmaking* para la computación *grid*. Buscad también información sobre la plataforma Open Science Grid.

2.5. Estandarización

El Open Grid Forum (OGF) es una comunidad de usuarios, desarrolladores y proveedores para la estandarización de la computación *grid*. Se formó en el año 2006 como fusión del Global Grid Forum y de la Enterprise Grid Alliance. El OGF tiene dos funciones principales (aparte de las funciones administrativas):

- Elaboración de normas relativas a las redes.
- Construcción de comunidades dentro de la red global (incluyendo su ampliación para alcanzar una participación más amplia de la academia y la industria).

Cada una de estas áreas funcionales se divide en grupos de tres tipos:

- Grupos de trabajo con un papel muy definido (en general, produce un estándar).
- Grupos de investigación con un papel menos significativo que reúne a la gente para discutir desarrollos dentro de su campo y generar casos de uso y organizar grupos de trabajo.
- Grupos de la comunidad.

Las principales normas que han sido producidas por la OGF son, entre muchas otras:

- GridFTP: ampliaciones en el protocolo de transferencia de archivos para la transferencia de datos a alta velocidad, segura y fiable.

- Grid Laboratory Uniform Environment (GLUE): es un modelo de información independiente de la tecnología para una representación uniforme de los recursos en *grid*.

- Single API for Grid Applications (SAGA): es una especificación de alto nivel para aplicaciones en red que describe una interfaz de programación de aplicaciones en *grid*.

- Open Grid Services Architecture (OGSA): describe una arquitectura para un servicio orientado al entorno informático de red para uso empresarial y científico.

- Distributed Resource Management Aplication API (DRMAA): es una especificación API de alto nivel para la presentación y el control de los trabajos en uno o más sistemas distribuidos de gestión de recursos (DRM) dentro de una arquitectura Grid.

- Envío de trabajos descripción del lenguaje: una especificación ampliable XML para la descripción de las tareas simples por no interactivas sistemas informáticos de ejecución. La especificación se centra en la descripción de las propuestas de tareas computacionales en los tradicionales sistemas de computación de alto rendimiento, como programadores de lotes.

2.6. Computación *grid* frente a supercomputación

Lo que realmente establece una barrera cualitativa entre la computación *grid* y la supercomputación es la heterogeneidad. Los nodos que conforman un *grid* pueden ser prácticamente cualquier cosa, desde un ordenador de sobremesa a un sensor del acelerador de partículas del CERN. Además, dentro del propia *grid* pueden convivir diferentes sistemas operativos y diferentes arquitecturas: desde los supercomputadores dedicados funcionando con un sistema tipo Unix a las agendas personales con Windows Mobile o Android del equipo directivo de la empresa. Los supercomputadores (que son clústeres por definición) se componen, en cambio, de hardware de características similares y el mismo sistema operativo.

Por otro lado, los nodos de un clúster se encuentran altamente acoplados, situados en el mismo dominio de red y habitualmente conectados con interfaces de red de altas prestaciones. Además, los nodos son dedicados y, por lo tanto, ejecutan únicamente las aplicaciones del clúster. En estas condiciones (conexiones rápidas, nodos dedicados, alto acoplamiento) lo más habitual es utilizar MPI para programar aplicaciones. De este modo, los clústeres son ideales para ejecutar aplicaciones subdivididas en muchos problemas pequeños y con gran cantidad de comunicación entre procesos, las aplicaciones paralelas de grano fino.

En cambio, en la computación *grid*, los nodos suelen estar distribuidos entre diferentes subredes y dominios, al encontrarse repartidos entre diferentes organizaciones. Además, es un entorno dinámico en el que podemos añadir o eliminar nodos en cualquier momento. Por otra parte, los nodos no son completamente dedicados, con el consecuente dinamismo a la hora de repartir las tareas. En el uso de máquina de red de gama baja (aprovechando los precios de la economía de escala) y la dispersión geográfica de sus nodos, las aplicaciones más apropiadas para ser ejecutadas en un Grid son las *coarse-grained*, aquellas

donde la comunicación entre procesos tiene poca relevancia frente a la cantidad de computación necesaria. Hay que señalar, sin embargo, que también hay herramientas para MPI que soportan computación *grid*, como MPICH-G2.

También cabe decir que aunque habitualmente la computación *grid* utiliza hardware de gran consumo, se pueden añadir computadores de altas prestaciones al *grid* y sumar así potencia de cálculo, o simplemente dar acceso a computación de altas prestaciones que solo está disponible en otras instituciones.

3. Computación *cloud*

En este apartado estudiaremos los fundamentos de la computación *cloud*. Primero nos centraremos en los conceptos básicos y aspectos generales de la computación *cloud* para, posteriormente, centrarnos en los aspectos relacionados con la computación de altas prestaciones. Hay que tener en cuenta que si bien la computación *grid* apareció como medio para poder mejorar la colaboración científica y poder así resolver problemas de gran relevancia, la computación *cloud* reutiliza gran parte de la experiencia obtenida con la computación *grid* con un componente principalmente económico, basándose en un modelo de mercado.

3.1. Introducción a la computación *cloud*

La computación *cloud*[30] se presenta como la materialización más cercana a la computación como utilidad[31]. Actualmente estamos viendo una gran transformación de la industria de las tecnologías de la información debido al modelo de computación *cloud*, que está basado en la idea de proveer software como servicio. Este modelo también está influenciando directamente en el modo como el hardware y los sistemas de tecnologías de la información se diseñan, se utilizan y se compran. Por ejemplo, el modelo de computación *cloud* permite que una pequeña empresa pueda empezar a operar sin tener que hacer grandes inversiones en hardware y los servicios asociados para operarlo, dado que puede comprar los servicios disponibles en el *cloud* solo cuando los necesite. Además, la asociatividad del coste de la computación *cloud* provoca que se puedan lograr soluciones escalables sin coste adicional. Por ejemplo, utilizar 1.000 servidores durante una hora cuesta lo mismo que utilizar uno durante 1.000 horas. Este modelo basado en la elasticidad de los recursos hace que la computación *cloud* tenga un gran potencial para revolucionar la computación en muchos ámbitos.

(30) En inglés, *cloud computing*.

(31) En inglés, *utility computing*.

Una de las definiciones más conocidas de la computación *cloud* es la proporcionada por el National Institute of Standards and Technology (NIST), U.S. Department of Commerce, que se presenta a continuación.

La **computación *cloud*** es un modelo que posibilita el acceso ubicuo, conveniente, bajo demanda a un conjunto de recursos configurables en red (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios), que se pueden proveer rápidamente y liberar con mínimo esfuerzo de gestión o de interacción del proveedor de servicios.

Este modelo *cloud* está compuesto por cinco características esenciales, tres modelos de servicio y cuatro modelos de despliegue.

Las características esenciales del modelo *cloud* son:

1) Autoservicio bajo demanda. Un consumidor puede ser provisto unilateralmente con recursos computacionales, como tiempos de servidor y almacenamiento en red, de manera automática y a medida que lo necesite, sin que sea necesaria la interacción humana de cada proveedor de servicios.

2) Amplio acceso a la red. Los recursos están disponibles en la red y se puede acceder a ellos mediante los mecanismos estándares que promueven el uso de plataformas heterogéneas de clientes ligeros o normales (por ejemplo, teléfonos móviles, tabletas, portátiles y estaciones de trabajo).

3) Agrupación de recursos. Los proveedores de los recursos computacionales se agrupan para servir a múltiples consumidores mediante un modelo multipréstamo, con diferentes recursos físicos y virtuales asignados y reasignados de manera dinámica según la demanda del consumidor. Hay un sentido de independencia de la ubicación en el que el cliente generalmente no tiene ningún control o conocimiento sobre la ubicación exacta de los recursos proporcionados, pero puede ser capaz de especificar la ubicación a un nivel más alto de abstracción (por ejemplo, país, estado o centro de datos). Algunos ejemplos de recursos son el almacenamiento, el procesamiento, la memoria y el ancho de banda de la red.

4) Rápida elasticidad. Los recursos pueden ser provistos y liberados elásticamente, en algunos casos de manera automática, para escalar rápidamente, ya sea creciendo o decreciendo, en función de la demanda. Desde el punto de vista del consumidor, los recursos disponibles a menudo parecen ser ilimitados y pueden ser asignados en cualquier cantidad en cualquier momento.

5) Servicio mesurado. Los sistemas *cloud* controlan y optimizan automáticamente el uso de los recursos mediante la capacidad de monitorización en un cierto nivel de abstracción adecuado para el tipo de servicio (por ejemplo, almacenamiento, procesamiento, ancho de banda y cuentas de usuario activas). El uso de recursos puede ser monitorizado, controlado y reportado, proporcionando transparencia tanto para el proveedor como para el consumidor del servicio utilizado.

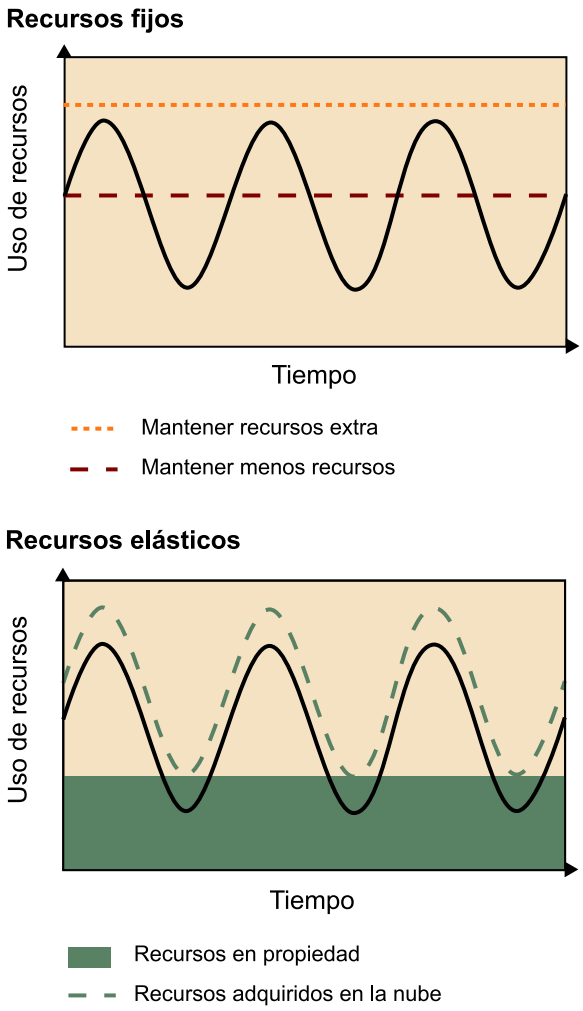
La definición del NIST también incluye tres modelos de servicio (SaaS, PaaS, IaaS) y cuatro modelos de despliegue o tipos de *cloud* (privado, de comunidad, público, híbrido), que veremos más adelante.

Desde el punto de vista del hardware existen tres aspectos nuevos en la computación *cloud*:

- La ilusión de infinitos recursos informáticos disponibles bajo demanda, eliminando así la necesidad de los usuarios del *cloud* de planear con anticipación la provisión de recursos necesarios.
- La eliminación de un compromiso por adelantado de los usuarios del *cloud*, lo que permite a las empresas empezar despacio y aumentar los recursos de hardware solo cuando aumentan sus necesidades.
- La posibilidad de pagar por el uso de los recursos informáticos (modelo *pay-as-you go*) a corto plazo, según sea necesario (por ejemplo, los procesadores por hora y el almacenamiento por día) y liberarlos cuando sea necesario, liberando de este modo máquinas y almacenamiento cuando ya no son útiles.

Cualquier aplicación necesita un modelo de cálculo, un modelo de almacenamiento y un modelo de comunicación. La multiplexación estadística necesaria para conseguir la elasticidad y la ilusión de infinita capacidad requiere que los recursos sean virtualizados para ocultar la implementación de la manera como se multiplexan y se comparte. Más adelante estudiaremos estos mecanismos de virtualización y sus soluciones más populares en la computación *cloud*.

Figura 11. Provisión de recursos *cloud* estática frente a elástica



Aunque el atractivo económico de la computación *cloud* es a menudo descrito como la conversión de los gastos de capital a gastos de explotación, el modelo de pago por uso capta mejor el beneficio económico para el comprador. La idea básica es que la ausencia de gastos de capital por adelantado permite que el capital se pueda invertir en negocio. Aun así, puede suceder que pagar por uso de recursos *cloud* parezca tener un precio similar al de comprar los recursos, pero el *cloud* permite transferir el riesgo a los proveedores, especialmente los riesgos de exceso de aprovisionamiento (subutilización) y de falta de aprovisionamiento (saturación).

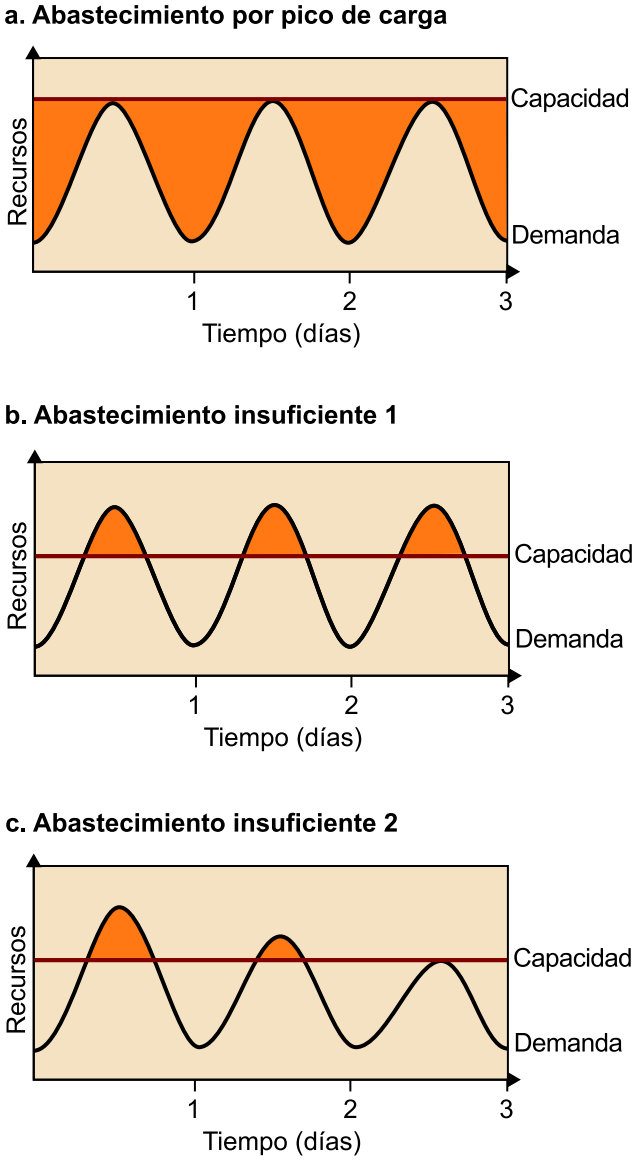
La elasticidad de los recursos es una de las claves, dado que permite que los usuarios puedan adaptar los recursos utilizados (y por los que pagan) bajo demanda, ya sea porque la utilización de los servicios tiene fluctuaciones periódicas (por ejemplo, más clientes durante el día que durante la noche), ya sea a causa de caídas inesperadas de la demanda por acontecimientos externos (por ejemplo, los casos de noticias), tal y como ilustra la figura 11.

Debemos tener en cuenta que adquirir nuevos equipos puede requerir semanas y el único modo de soportar estos picos en la demanda es el suministro anticipado. Esto provoca que, si no se dispone de los recursos para soportar los picos de demanda, se pierda capacidad, tal como muestra la figura 12a. También puede suceder que se infravalore el pico de demanda (figura 12b) y no se disponga de suficientes recursos para dar servicio a los clientes. Mientras que los efectos monetarios de exceso de aprovisionamiento son fáciles de medir, los de falta de aprovisionamiento son más difíciles de medir pero potencialmente igual de graves: no solo se rechazan usuarios, por lo que no se generará ningún ingreso, sino que muchos usuarios se perderán para siempre a causa de un mal servicio. La figura 12c tiene como objetivo captar este comportamiento: los usuarios abandonan un servicio que está saturado hasta que la carga pico es igual a la capacidad de uso del centro de datos, en el que los usuarios reciben de nuevo un punto de servicio aceptable, pero con un menor número de usuarios potenciales.

Uno de los pioneros de la computación *cloud* fue Salesforce.com, que en 1999 introdujo el concepto de entrega de aplicaciones empresariales a través de una web básica. Amazon fue la siguiente en elegir Amazon Web Service (2002). Entonces apareció Google Docs (2006), que puso la computación *cloud* a la vanguardia de la conciencia del público. Elastic Compute Cloud de Amazon (Amazon EC2) surgió en el mismo año como un servicio web comercial que permitió a empresas y particulares alquilar equipos en los que se pudieran ejecutar sus propias aplicaciones informáticas. En el año 2008 surgió Eucalyptus, la primera plataforma de código abierto compatible con las interfaces de Amazon Web Services para poder enlazar los *clouds* privados con el proveedor de servicios *cloud* más popular. Algo más tarde apareció OpenNebula, el primer software de código abierto para la implementación de los *clouds* privados e híbridos. Windows Azure, la plataforma *cloud* de Microsoft, se dio a conocer en el 2009. Un año más tarde, las plataformas adoptaron una nueva estructura

propia con las evoluciones de diferentes capas de servicio: cliente, aplicación, plataforma, infraestructura y servidor. En el 2011, Apple se unió al club de proveedores de servicios *cloud* con su servicio iCloud, un sistema de almacenamiento en el *cloud* orientado a música, vídeos, fotografías, aplicaciones y calendarios.

Figura 12. Ejemplificación de varios escenarios de provisión de recursos



3.2. Características de la computación *cloud*

En este subapartado nos centraremos en las características generales de la computación *cloud* que están más orientadas a entornos empresariales y modelos de negocio que a computación de altas prestaciones. Más adelante estudiaremos las implicaciones y las posibilidades de la computación *cloud* para la computación de altas prestaciones.

El modelo de computación *cloud* tiene una serie de características que se pueden considerar como ventajas comunes para la mayoría de las empresas, por ejemplo:

- **Flexibilidad**: las empresas pueden contratar lo que necesitan en cada momento, dado que los requisitos de una empresa pueden depender de muchos factores y por lo tanto, ser muy variables. En un sistema tradicional, si por ejemplo se requería un sistema con mucha capacidad de computación durante un periodo reducido, no se podía amortizar su coste.

- **Coste**: el modelo de pago por uso permite a las compañías reducir los costes fijos y las inversiones en tecnologías de la información, además de aprovechar al máximo su dinero, ya que pueden pagar exactamente por lo que necesitan. El precio de este tipo de servicios es competitivo gracias a la explotación de las economías de escala.

- **Subcontratación**: el *cloud* permite externalizar responsabilidades relacionadas con la gestión de las tecnologías de la información, reduciendo así el personal que se necesita para llevar a cabo estas tareas.

- **Gestión de versiones**: el cliente no ha de preocuparse de actualizar su infraestructura o comprar licencias de software más nuevas. El proveedor del servicio se encarga de la actualización de las licencias con los costes asociados.

- **Seguridad**: el proveedor es el encargado de garantizar la seguridad del sistema y los datos, evitando que elementos externos u otros usuarios puedan acceder a él, aunque físicamente estén usando los mismos servidores. Además, se pueden ofrecer servicios de copias de seguridad de un modo transparente para evitar las pérdidas de datos en caso de fallo.

- **Disponibilidad**: el proveedor debe garantizar que el servicio esté en funcionamiento durante el tiempo contratado. Este aspecto es crítico, dado que la actividad de una empresa no se puede detener por problemas informáticos. Para garantizar esto con el sistema tradicional habría que tener duplicidad de los elementos del sistema.

Estas características del modelo de computación *cloud* tienen una serie de **ventajas** respecto a los modelos tradicionales de computación, como:

- Presentación de los servicios a escala global: gracias a la replicación de servicios en diferentes lugares del mundo se puede hacer accesible la información con muy buena calidad de trabajo.

- Las infraestructuras de computación *cloud* proporcionan una mejor capacidad de adaptación, recuperación completa de pérdida de datos mediante las copias de seguridad y acceso a la información en cualquier momento.

- A un usuario que utilice una infraestructura con 100% de computación *cloud* no le debe preocupar el hardware que hay ni su mantenimiento; todo esto pasa a manos del proveedor del servicio, lo que requiere mucha menos inversión en hardware y software y permite empezar a trabajar más pronto.

- Las implementaciones de las aplicaciones suelen ser mucho más rápidas, dado que muchas están hechas mediante sistemas base que permiten un cierto nivel de personalización.

- Las actualizaciones del software no hay que hacerlas en las máquinas físicas, sino que se hacen automáticamente en el *cloud*. De este modo no se pierde tiempo en actualizaciones.

- Ayuda a disminuir el consumo energético, dado que los centros de datos de las empresas no deben mantener sus servidores en marcha porque todo está externalizado.

La computación *cloud* también tiene una serie de **inconvenientes** que hay que tener en cuenta antes de adoptar soluciones *cloud*, como:

- La centralización de las aplicaciones y el almacenamiento de datos provoca una dependencia del proveedor de servicios.

- La disponibilidad de los servicios y de los datos está vinculada a tener acceso a Internet; por lo tanto, si hubiera problemas de conexión no se podría acceder a ellos.

- Los datos sensibles no son locales, lo que podría infringir parte de la ley de protección de datos, dado que exponen los datos a la posibilidad de robo de información.

- La fiabilidad del servicio recae sobre la tecnología que utilizan los proveedores. Muchas veces el cliente no sabe qué sistemas son los que dan servicio y no sabe si podrían ser mejores por el mismo precio.

- La seguridad es también un punto clave; la información debe pasar por diferentes nodos para llegar a su destino y en cada uno de estos nodos pueden existir vulnerabilidades.

3.3. Tipos de *clouds*

Existen diferentes tipos de *clouds* (o modelos de despliegue) que hay que detallar porque cada uno tiene unas características diferentes, lo que provoca que unos se ajusten mejor que otros según el tipo de cliente o empresa.

3.3.1. *Clouds* públicos

El tipo de ***cloud* público** consiste en un proveedor que ofrece servicios de computación a través de Internet de manera abierta a todo el mundo que quiera comprar sus servicios.

Las **características** de este tipo de *clouds* son:

- **Disponibilidad**: a partir de que se contrata el servicio, en un breve lapso de tiempo ya está disponible. Además, el proveedor se encarga de que una vez contratado este se halle disponible durante todo ese periodo de tiempo.

- **Pago por uso**: permite contratar solo aquellos recursos que se necesitan, evitando así la necesidad de invertir en una infraestructura para suplir esta necesidad.

- **Variedad de servicios**: permiten externalizar todas las funciones básicas del usuario.

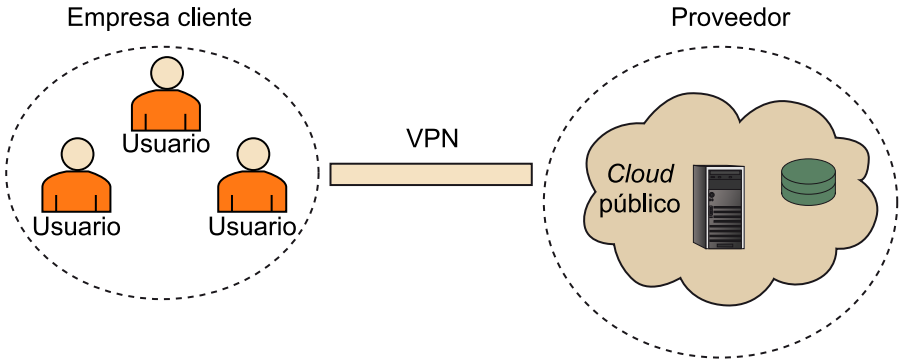
- **Seguridad**: el proveedor es el encargado de cumplir los requisitos de seguridad y protección de datos.

- **Mantenimiento**: el proveedor es el encargado de mantener la arquitectura necesaria para proporcionar el servicio contratado.

- **Escalabilidad**: permite aumentar o reducir los servicios según las necesidades del cliente a lo largo del tiempo y con un periodo de implementación corto.

La figura 13 ejemplifica la configuración de *cloud* público.

Figura 13. Esquema de *cloud* público genérico



3.3.2. *Clouds* privados

En el tipo de ***cloud* privado**, el propio cliente gestiona los recursos, que se pueden encontrar físicamente en sus propias instalaciones o no.

Las **características** de este tipo de *clouds* son:

- **Reutilización**: permite aprovechar inversiones de hardware y personal realizadas anteriormente.

- **Mayor tiempo de implantación que el *cloud* público**: preparar la infraestructura de *cloud* y redimensionarla, dependiendo de las necesidades, requiere más tiempo que en el caso del *cloud* público, dado que el propio usuario debe encargarse de realizar estas gestiones.

- **Más especificidad**: el *cloud* se puede diseñar ajustándose más a las necesidades concretas del usuario.

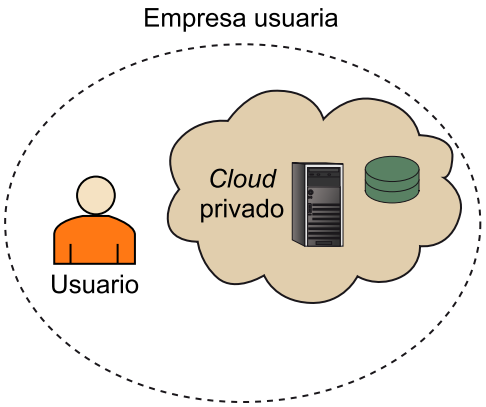
- **Gestión propia de los sistemas y los datos**: esto implica que la empresa deberá encargarse del mantenimiento y la seguridad.

- **Ancho de banda**: al poder situar el *cloud* dentro de la propia organización, se eliminan las restricciones de ancho de banda.

- **Seguridad**: si el *cloud* es solo de uso interno del usuario, se puede mantener aislado de Internet, evitando así posibles ataques externos. También se obtiene mayor control sobre la gestión de la seguridad; el *cloud* público depende de la gestión que haga de él el proveedor.

La siguiente figura ejemplifica la configuración de *cloud* privado.

Figura 14. Esquema de *cloud* privado genérico



3.3.3. *Clouds* de comunidad

En el tipo de ***cloud* de comunidad**, el *cloud* es controlado y utilizado por un grupo de organizaciones que tienen intereses compartidos, como los requisitos específicos de seguridad o una función común. Los miembros de la comunidad comparten el acceso a los datos y aplicaciones en el *cloud*.

Este tipo *clouds* tienen unas **características** intermedias entre lo público y lo privado:

- **Mayor cantidad de recursos** que con un *cloud* privado, aunque generalmente menos que con un *cloud* público.
- **Especificidad**: el diseño del *cloud* puede estar más ajustado a unas necesidades concretas, dado que las organizaciones que lo forman tienen puntos en común.
- **Seguridad**: en términos de seguridad, tiene la ventaja respecto a los *clouds* públicos de ser de acceso más restringido. Aun así, es más abierto que los privados.
- **Menor gestión de la infraestructura** y los datos que en un privado pero mayor que en un público.
- Requiere **inversión** en hardware y software.

3.3.4. *Clouds* híbridos

El tipo de ***cloud* híbrido** se caracteriza por unir dos o más tipos de *clouds* diferentes.

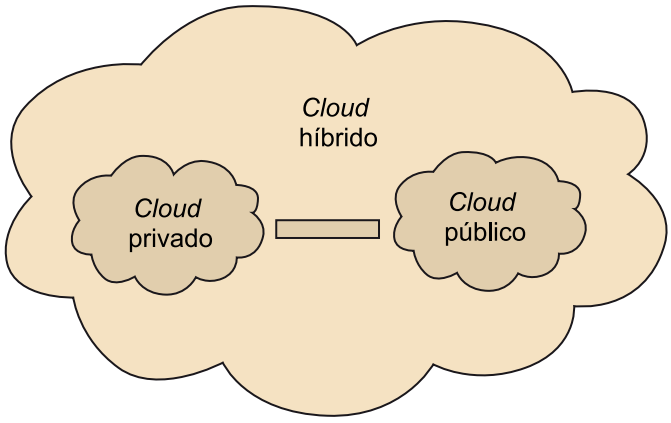
Estos *clouds* siguen siendo entidades únicas pero interconectadas entre sí, lo que permite disfrutar de las ventajas de los diferentes tipos de *clouds*:

- **Complejidad**: al unir diferentes tipos de *clouds* la complejidad es mayor, dado que hay que integrar los diferentes *clouds* con sus tecnologías asociadas.
- **Flexibilidad y control**: permite una mayor flexibilidad a la hora de prestar servicios, así como un mayor control de estos y de los datos.
- Permite utilizar cada tipo de *cloud* para los servicios que **se adapten mejor** a las características de cada uno.

Normalmente los *clouds* híbridos suelen intentar explotar primero los recursos propios (para evitar pagar por uso) y solo cuando el usuario no tiene suficiente capacidad con los recursos locales, utiliza el *cloud* público (por ejemplo, como extensión o forma de aceleración). Este tipo de técnica se suele denominar *cloud bursting*.

La siguiente figura ejemplifica la configuración de *cloud* híbrido.

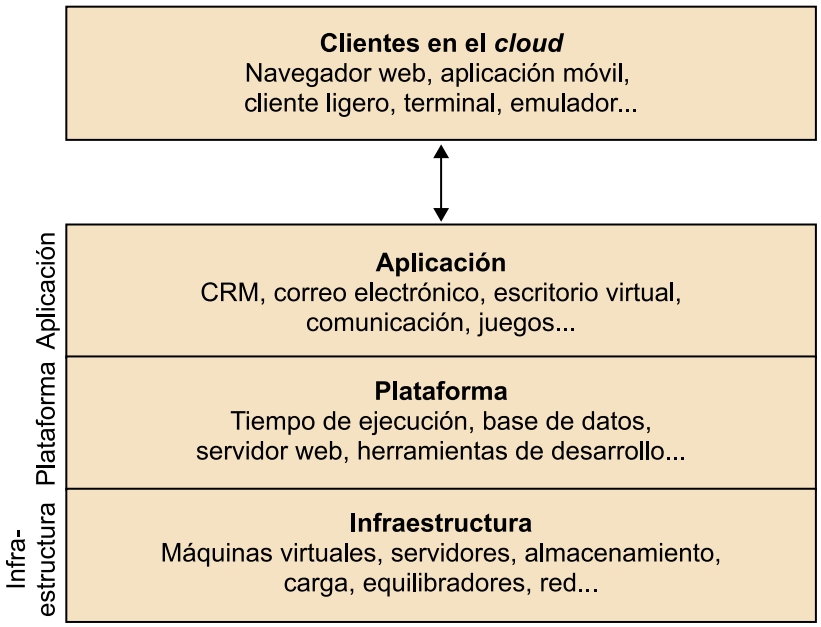
Figura 15. Esquema de *cloud* híbrido genérico



3.4. Modelos de servicio

Tal y como se ha comentado, existen tres modelos básicos de provisión de servicios *cloud*. Estos modelos también se suelen ver como una jerarquía por capas, en la que las capas superiores son más abstractas y cercanas al usuario (por ejemplo, el software) y las capas inferiores están más cercanas del hardware, como muestra la figura 16.

Figura 16. Modelo por capas de la computación *cloud*



3.4.1. Infraestructura como servicio (IaaS)

En el modelo o capa de infraestructura como servicio (IaaS)[32], el proveedor ofrece un servicio de computadoras, máquinas virtuales, que pone a disposición del cliente.

[32]Del inglés *infrastructure as a service*.

Por lo tanto, el servicio consiste en toda la parte de hardware y sistema operativo que el cliente debe gestionar según sus intereses. El proveedor solo se ha de preocupar de mantener estas máquinas en funcionamiento y conectadas a Internet, delegando toda la gestión de software (sistema operativo, bases de datos, servidores de aplicaciones) al usuario.

3.4.2. Plataforma como servicio (PaaS)

En el modelo o capa de plataforma como servicio (PaaS)[33], el servicio va un paso más allá del de infraestructura como servicio y, en este modelo, el cliente recibe la plataforma entera (sistema operativo, base de datos, servidor de aplicaciones, lenguaje de programación, entre otras) como servicio.

[33]Del inglés *platform as a service.*

En este tipo de servicio el cliente solo se debe preocupar de gestionar las aplicaciones que quiere emplear en la plataforma y no es necesario que gestione conceptos relacionados con la gestión ni el mantenimiento de la plataforma o de la infraestructura.

3.4.3. Software como servicio (SaaS)

El modelo o capa de software como servicio (SaaS)[34] es el modelo preferiblemente recomendado para un cliente final y consiste en la adquisición del servicio a más alto nivel, que corresponde al nivel de aplicación.

(34)Del inglés *software as a service.*

En este servicio el cliente solo se ha de preocupar de gestionar las configuraciones y/o parámetros de la aplicación ofrecida por el servicio, dejando de lado tanto la máquina donde reside esta como el sistema operativo que ha instalado u otros detalles que no son relevantes para el usuario, quien no tiene por qué tener ningún conocimiento más allá de la utilización del software.

Tal y como se muestra en la figura 16, cada uno de los modelos o capas necesita a su predecesor para poder ofrecer este servicio. Así pues, si un proveedor quiere ofrecer un software como servicio (SaaS), debe disponer de una plataforma como servicio (PaaS), o en otro caso debe contratar este servicio a un tercero. Lo mismo con la capa de infraestructura como servicio, que es el modelo de menor nivel. Este hecho provoca que las vulnerabilidades y los agujeros de seguridad de los modelos de computación *cloud* deban dividirse según su modelo y se gestionen según este, ya sean el proveedor o el cliente los encargados de hacerlo. En este sentido, como clientes la ventaja de disponer de un servicio de más alto nivel (SaaS) ofrece la desventaja de no poder controlar la gestión de la seguridad de este y a la inversa. Por ejemplo, si disponemos de un software como servicio (SaaS), perdemos el control de la gestión del sistema operativo, actualizaciones, requisitos de acceso a los datos y privacidad, entre otros. Mientras que si disponemos de un servicio de infraestructura, seremos nosotros mismos quienes deberemos gestionar toda la seguridad tanto a nivel de sistema operativo como de aplicaciones y software.

También hay otros modelos de computación *cloud* que nacen a partir de conjunciones de los tres modelos principales que hemos visto. Los más relevantes son:

- Red como servicio (NaaS)[35].
- Almacenamiento como servicio (STaaS)[36].
- Datos como servicio (DaaS)[37].
- Entorno de pruebas como servicio (TeaaS)[38].

(35)Del inglés *network as a service.*

(36)Del inglés *storage as a service.*

(37)Del inglés *data as a service.*

(38)Del inglés *test environment as a service.*

También existen otros más especializados, como los relacionados con la seguridad, la monitorización de múltiples aspectos, etc.

Actividad

Buscad información sobre diferentes proveedores de servicio *cloud* y clasificadlos en función del tipo de modelo de servicio que ofrecen.

3.5. Casos de uso

A partir de lo que hemos visto anteriormente, en este subapartado se ha realizado una selección de los que se consideran posibles casos de uso que puedan provocar que un cliente/empresa opte por un sistema *cloud*:

1) Adaptarse a las variaciones estacionales de carga: Las empresas, dependiendo de su sector o sistema de funcionamiento, como el caso de empresas que solo trabajan durante ciertos periodos del año o tienen picos importantes de trabajo, necesitan tecnologías de la información muy variadas y difíciles de planificar. Este caso de uso es muy importante, dado que utilizar un *cloud* público permite redimensionar la capacidad de los sistemas de tecnologías de la información para adaptarse a las variaciones de carga o a la necesidad de nuevos servicios. La capacidad del *cloud* para este caso de uso viene dada gracias a la virtualización y a la subcontratación[39]. La virtualización permite aumentar o disminuir los recursos de hardware o servicios contratados por cada compañía de manera ágil, gracias a que esta proporciona independencia del hardware real.

[39] En inglés, *outsorcing*.

2) Cambio en el sistema de financiación: Otro caso de uso popular de la computación *cloud* es aprovecharlo para pasar de un sistema de financiación consistente en comprar activos fijos para conseguir realizar las tareas de tecnologías de la información necesarias, a pagar por servicios. De este modo se obtienen varios beneficios:

- Se reduce el riesgo, dado que toda inversión implica un riesgo. La situación de una compañía siempre puede variar, ya sea debido a un elemento externo o interno. Además, no se puede garantizar que esta inversión se pueda amortizar. Por ello se opta por el pago de servicios, porque estos son costes a corto plazo y pueden anularse o modificarse si en un futuro se cree necesario.

- Conseguir ventaja competitiva. En empresas pequeñas o medianas, el *cloud* permite el acceso a las empresas a recursos que de otra manera habrían estado fuera de su alcance debido al gran coste que pueden tener.

- Como el *cloud* todavía está en un proceso de implantación, las empresas optan por adherirse a él antes que su competencia para poder aportar un valor mayor que las otras. Este valor lo pueden conseguir, por ejemplo, obteniendo mejores herramientas de gestión que les permitan mejorar su producto o servicio, o consiguiendo herramientas que les permitan un mejor contacto con los clientes.

3) Externalización de las tecnologías de la información: La externalización de los servicios de tecnologías de la información ofrece una gran cantidad de ventajas a pesar de que no está exenta de algunos inconvenientes, por eso algunas empresas optan por soluciones de *cloud* privado o híbrido. Las principales ventajas de la externalización son:

- Reducción del personal necesario para el mantenimiento, aunque este también puede ser externo en el sistema tradicional.

- Espacio físico de los elementos de hardware: el usuario de *cloud* público no necesita un espacio físico para mantener los servidores porque estos son virtualizaciones de los servidores del proveedor.

- Se evita el mantenimiento tanto de hardware como de software, dado que el proveedor se encarga de mantener los servicios actualizados y funcionando.

4) Proyectos con un corto ciclo de vida: En las empresas que realizan proyectos, especialmente si estos son cortos y con variedad de requisitos de aplicaciones y de capacidad de computación, el hecho de adquirir todo lo necesario para su realización aumentaría notablemente el coste de desarrollo del proyecto, dado que con el breve tiempo de desarrollo no se podría amortizar una inversión en lo que fuera necesario. Si una empresa realiza este tipo de proyectos, le resultará muy beneficiosa un *cloud* público, ya sea en conjunción con un privado o solo con el público. De este modo pueden calcular fácilmente el coste del alquiler de los servicios necesarios para el desarrollo, durante el tiempo estimado, y añadirlos al presupuesto. Asimismo, esto les permite un presupuesto global menor y una mayor agilidad en los proyectos, al minimizar la implantación de los entornos necesarios.

5) Recuperación de desastres: Después de un desastre, por ejemplo natural, que haya dañado el sistema informático de una empresa, para poder retomar su funcionamiento con la máxima celeridad posible mientras se restablece el sistema de la empresa, se utiliza computación *cloud*. También se utiliza en países en desarrollo que tienen dificultades para acceder a este tipo de infraestructuras o servicios de manera tradicional.

6) Necesidad de computación escalable masiva: Empresas que requieren una capacidad de procesamiento muy grande pueden optar por utilizar un servicio IaaS solicitando la capacidad de computación necesaria.

7) Consolidación de infraestructura: Necesidad de una infraestructura de tecnologías de la información sólida capaz de soportar todas las necesidades de negocio actuales y con flexibilidad para adaptarse a futuros cambios.

8) Acceso universal: Otro elemento importante del *cloud* es la posibilidad de acceder a los servicios del negocio desde diferentes localizaciones y dispositivos. Esta función permite tanto unificar con mayor facilidad las tecnologías de la información de diferentes sedes en localizaciones separadas –consiguiendo que todos puedan acceder a los mismos recursos de tecnologías de la información como si se tratara de una única sede–, como la capacidad de controlar, modificar o utilizar elementos de las tecnologías de la información desde fuera de la empresa con diferentes dispositivos, como móviles o tabletas. Esta función puede ser muy útil para empresas que se dediquen a negocios en tiempo real, como por ejemplo, la compraventa de acciones. Pueden realizar operaciones aunque no se encuentren en su puesto de trabajo, o tareas más simples, como consultar el correo electrónico de la empresa desde fuera de la intranet.

9) Seguridad: Protección de datos y copias de seguridad. Las empresas pueden preferir utilizar un proveedor que les garantice la seguridad y la protección de los datos antes que implementar el que tienen. Esto se debe a que algunos proveedores tendrán más experiencia en estos temas que la propia empresa. Además de la protección de los datos contra ataques, hay que valorar que también se ofrece un sistema de réplica de los datos para evitar pérdidas. Para la propia empresa, tener un sistema de copias de seguridad propio supondría un coste importante en hardware y en la gestión de estos.

10) Rápida implantación: La rápida implantación se da principalmente en los casos de *cloud* público y permite a nuevas empresas iniciar su proceso de negocio con un tiempo menor que si utilizan un sistema de tecnologías de la información tradicional. También permite más flexibilidad a la hora de realizar nuevos proyectos, implementar nuevos servicios o pasar de la infraestructura disponible al *cloud*.

11) Disminución del *time to market*: En proyectos como desarrollos informáticos, permite acortar el tiempo entre las diferentes fases de desarrollo y test, facilitando la adquisición de herramientas y la preparación de entornos para realizar el proceso. En caso contrario, estos cambios de fase podrían ser mucho más costosos. Por último, en algunos casos también puede facilitar la distribución, si esta se distribuye usando el propio *cloud*.

12) Sistema de tecnologías de la información compartido con empresas con intereses similares: Los *clouds* compartidos presentados en el subapartado anterior también suponen un motivo importante para decantarse por el *cloud*, dado que conjuntos de empresas con intereses comunes pueden compartir el *cloud*. Esto permite crear un *cloud* más específico que uno público, dado que se puede invertir en crear uno que se ajuste lo máximo posible a las necesidades del conjunto. Al compartir el *cloud*, también facilita la compartición de datos y una mayor capacidad de cooperación.

13) Reducción de emisiones peligrosas para el medio ambiente: Las empresas con gran cantidad de hardware pueden reducir de manera drástica su consumo energético, reduciendo así la contaminación que se provoca para obtener este tipo de energía. Esta solución es ficticia si el proveedor no utiliza energías renovables, dado que reduce el consumo de la empresa pero se aumenta el del proveedor.

3.6. Virtualización

Tal y como se ha comentado anteriormente, la virtualización es una de las claves para la computación *cloud* puesto que permite la multiplexación necesaria para conseguir elasticidad y la ilusión de infinita capacidad. A continuación nos centraremos en los conceptos más básicos de virtualización y posteriormente nos centraremos en las implicaciones que tienen en la computación *cloud*.

La **virtualización**, en un sentido amplio, consiste en una capa de abstracción de software que se añade entre el hardware y el sistema operativo que se ejecuta en el sistema. La capa resultante tiene las siguientes funciones:

1) Permite que múltiples sistemas operativos se ejecuten simultáneamente en una única máquina física.

2) Permite particionar dinámicamente y aislar la disposición de los recursos físicos, como la memoria, la/las CPU, los discos y los dispositivos de entrada/salida.

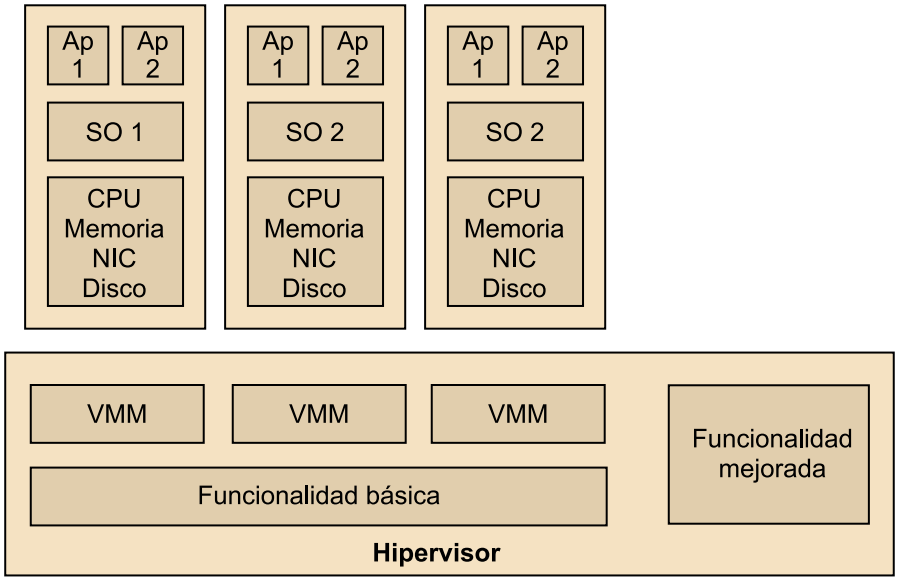
La virtualización es una tecnología muy potente que ofrece grandes beneficios, como permitir la consolidación de servidores proporcionando una óptima utilización de los recursos, lo que simplifica el desarrollo de software y la realización de pruebas, y la mejora de la agilidad en los centros de datos.

Observad que en este subapartado haremos especial énfasis en la virtualización x86. La figura 17 muestra la estructura básica de un sistema virtualizado. La capa de virtualización de base es el software que organiza y gestiona diferentes máquinas virtuales en los monitores de máquinas virtuales (VMM)[40]. El hipervisor es el componente encargado de la interacción directamente con el hardware. Su funcionalidad es diferente para diferentes arquitecturas e implementaciones. Cada VMM se ejecuta en el hipervisor y tiene su propia abstracción de máquina virtual encargada de ejecutar un sistema operativo invitado[41]. Sus funciones principales son la partición y compartición de los diferentes recursos, como los dispositivos de CPU, memoria, disco y entrada/salida, de manera que todo el sistema es compartido por multiplexación entre diferentes máquinas virtuales.

[40]Del inglés *virtual machine monitor*.

[41]En inglés, *guest operating system*.

Figura 17. Estructura básica de un sistema virtualizado



3.6.1. Tipos de virtualización

La virtualización de la arquitectura x86 implica que el sistema de virtualización pueda trabajar directamente con el hardware, de modo que necesitan que el sistema tenga un control total de los recursos físicos.

La figura 18 muestra cuatro niveles diferentes de privilegios inherentes a x86, conocidos como anillo 0, 1, 2 y 3, que denotan el nivel de acceso al hardware del ordenador. El anillo 0 se usa principalmente para las aplicaciones de usuario, el 3 para el sistema operativo con funcionalidades privilegiadas que utilizan directamente los recursos del sistema, y los anillos 1 y 2 rara vez son utilizados.

Los tipos de virtualización también se pueden clasificar en las siguientes categorías:

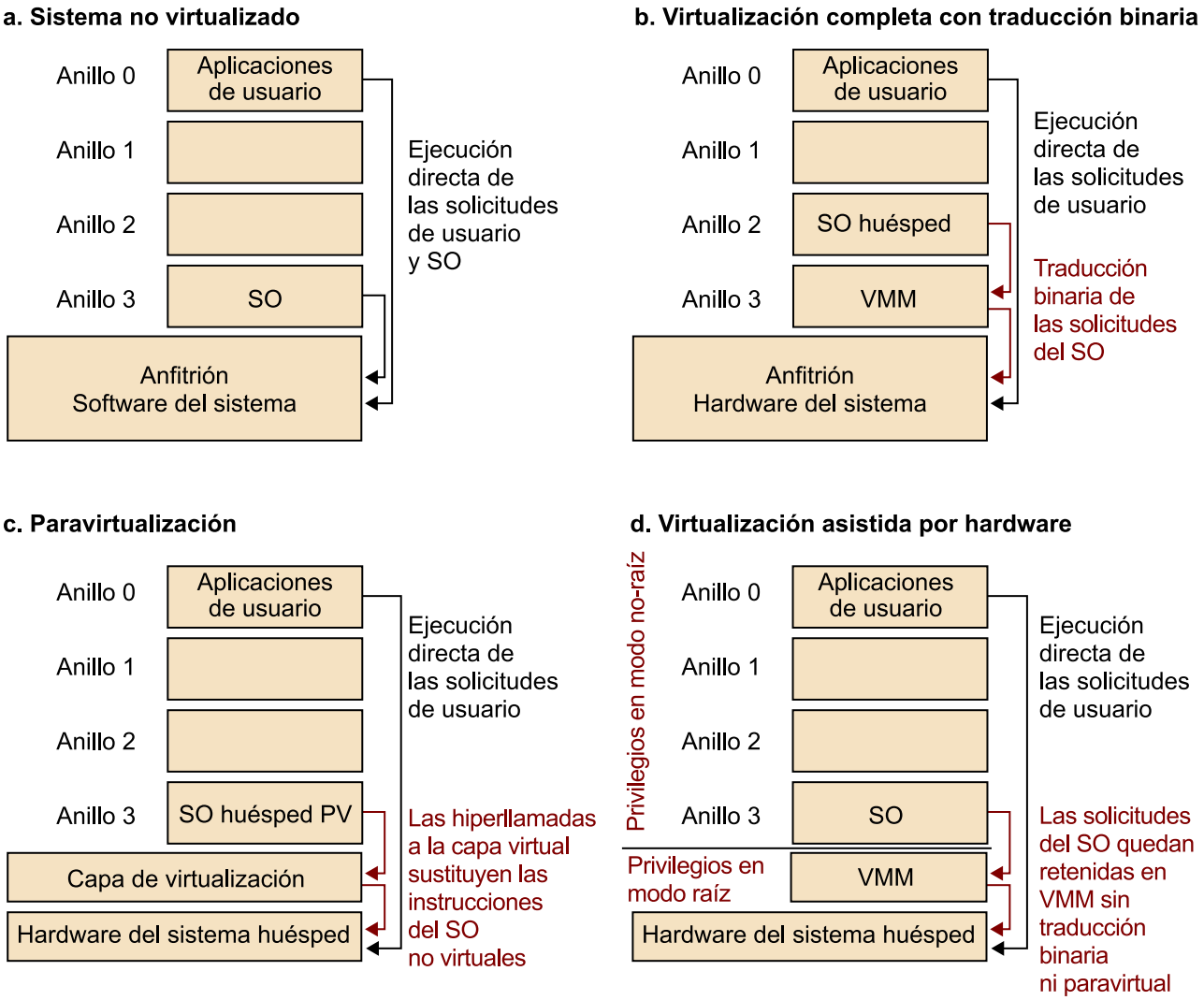
- Virtualización CPU.
- Virtualización de memoria y dispositivo.
- Virtualización de entrada/salida.

La virtualización de CPU se puede dividir en tres subtipos en función de los mecanismos utilizados para gestionar las instrucciones privilegiadas y la flexibilidad para lograrlo. Estos son:

- Virtualización completa con traducción binaria.
- Paravirtualización.
- Virtualización asistida por hardware.

La figura 18 muestra un sistema no virtualizado y con diferentes técnicas para conseguir virtualización mediante los anillos de privilegios.

Figura 18



La virtualización completa con traducción binaria consiste en una combinación de traducción binaria (traducción de código del núcleo del sistema operativo para reemplazar código no virtualizable con un conjunto de instrucciones que tengan el mismo efecto) con ejecución directa (ejecución de código de usuario directamente en el procesador), de modo que cada VMM proporciona un conjunto de BIOS virtuales, dispositivos y módulos de gestión de memoria. Dado que el hipervisor traduce las instrucciones del sistema operativo en tiempo de ejecución, dejando el código de usuario sin modificar, el sistema operativo es independiente del hardware subyacente. Algunos de los beneficios obtenidos por esta técnica incluyen un completo aislamiento de alta seguridad, portabilidad y migración eficiente de las máquinas virtuales. Ejemplos de este tipo de virtualización son los productos de la empresa VMWare y MS Virtual Server.

La paravirtualización requiere modificaciones en el núcleo del sistema invitado, reemplazando instrucciones no virtualizables con hiperllamadas que se comunican directamente con el hipervisor para realizar operaciones críticas, como la gestión de interrupciones. Aunque no tiene prestaciones de virtuali-

zación completa, la sobrecarga en cuanto a rendimiento es baja y variable en función de la carga de trabajo. Un ejemplo de este tipo de virtualización es Xen.

La virtualización de memoria consiste en la compartición y asignación dinámica de memoria física a las máquinas virtuales. Del mismo modo que en la gestión de memoria proporcionada por los sistemas operativos modernos, hay un nivel adicional de memoria debido a la virtualización de la MMU. El VMM utiliza hardware TLB para mantener las tablas de páginas para una búsqueda directa durante el mapeo del sistema operativo invitado en la memoria física.

En la virtualización de dispositivo y de entrada/salida gestiona el enrutamiento de solicitudes de entrada/salida entre los dispositivos virtuales y el hardware físico compartido, que permite una gran cantidad de prestaciones con una administración sencilla. Los dispositivos virtuales pueden interactuar entre sí sin problemas y sin afectar a los recursos físicos, y se pueden migrar o manipular de manera espontánea. El hipervisor estandariza todos los dispositivos virtuales que permitan la portabilidad entre plataformas gracias a una configuración compatible de hardware virtual en cualquier hardware físico.

La virtualización ofrece varias **ventajas** respecto a los sistemas tradicionales, entre los que encontramos:

- **Facilidad de depuración**. Los desarrolladores pueden probar nuevos sistemas operativos en máquinas estables y también pueden depurar su código con más eficacia debido al aislamiento de los diferentes sistemas. La recuperación también es más rápida mediante el uso de versiones guardadas de los clientes.

- **Balanceo de la carga y reubicación**. Las máquinas virtuales se pueden migrar entre equipos para equilibrar la carga de trabajo. También permiten agregar diferentes recursos para mejorar la eficiencia.

- **Consolidación de servidores**. Una única máquina física puede ejecutar muchas máquinas virtuales. Cada máquina virtual tiene su propio acceso a la raíz y puede elegir los tipos de aplicaciones/servicios que se ejecuten sin depender de otros usuarios. Esto también provoca que se maximice la utilización de los sistemas multiprocesador. El arranque de un sistema operativo resulta tan simple como iniciar una aplicación.

- **Menor coste de propiedad**. La utilización óptima de los recursos físicos reduce el coste total de propiedad de las empresas, así como los costes de formación de los empleados gracias a la reconfiguración mínima de los sistemas. Los usuarios pueden ejecutar productos heredados en un entorno seguro en nuevas arquitecturas ahorrando en los costes de refinamiento de la aplicación.

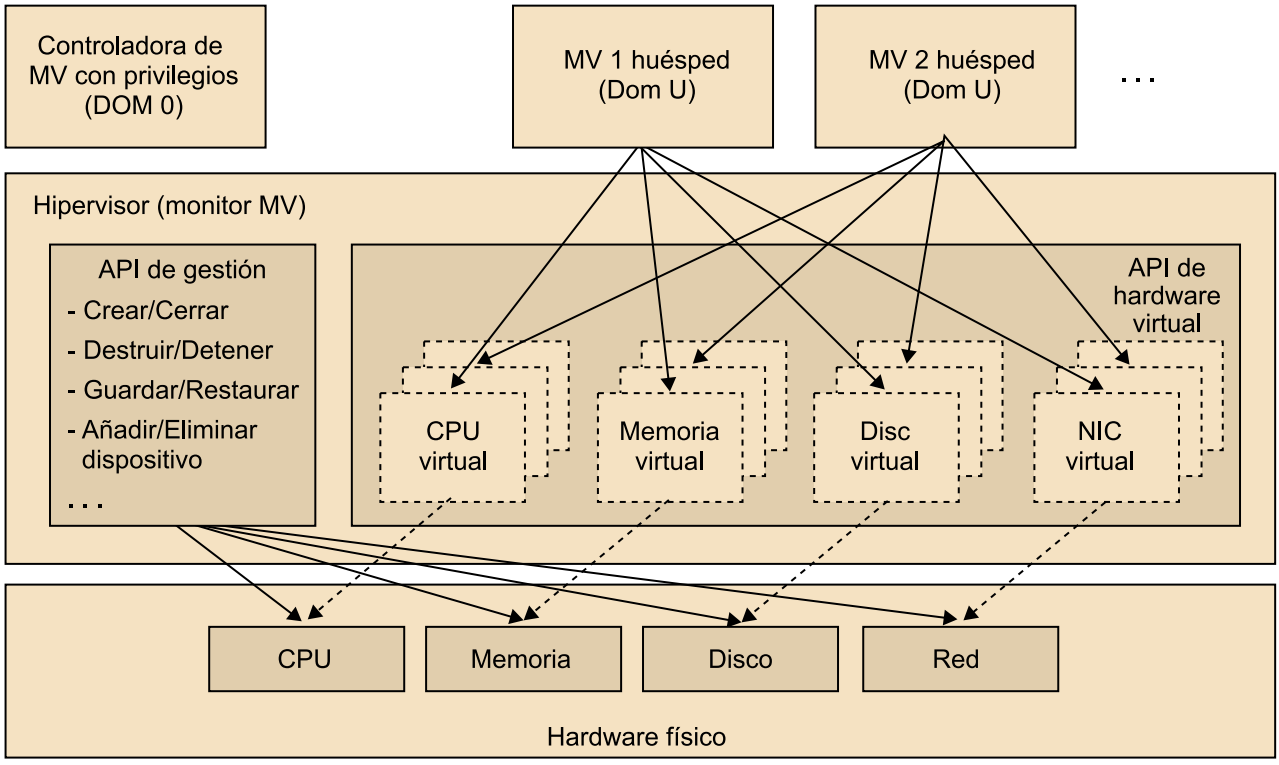
- **Menor consumo de energía e infraestructura de refrigeración**. La utilización eficiente de recursos se traduce en un uso óptimo de la energía y la reducción de sus costes. Disponer de una infraestructura más reducida nos permite una eficaz refrigeración de los centros de datos con menos sistemas de aire acondicionado, lo que reduce los costes energéticos.

En este subapartado utilizamos Xen como caso de uso de sistema de virtualización, ya que es uno de los más utilizados (incluyendo *clouds* públicos como Amazon EC2).

La arquitectura de Xen consta de tres componentes principales por encima del hardware, tal como muestra la figura 19. Está compuesta por tres componentes básicos:

- Hipervisor Xen/monitor de máquinas virtuales.
- Dominio privilegiado/conductor (Dom 0 en la figura).
- Dominios de usuario/invitados (Dom U en la figura).

Figura 19. Esquema general de la arquitectura Xen



El hipervisor o monitor de máquina virtual (VMM) de Xen es el componente que tiene relación directa con el hardware físico y que proporciona una interfaz virtual a los dominios que Xen aloja. Sus **funciones básicas** son las siguientes:

- Cada dominio recibe una porción específica de los recursos de la máquina física. Esta distribución de recursos puede ser arbitraria, equitativa o seguir alguna directiva de usuario. El VMM puede optar por restringir el acceso a

algún dispositivo físico de un dominio invitado o incluso puede crear un dispositivo virtual que no existe.

- El VMM ofrece dispositivos virtuales a los dominios. La fabricación de un dispositivo es de importancia secundaria, dado que se trata de un dispositivo de tipo general, un dispositivo de red o un dispositivo de bloques en caso de dispositivos de almacenamiento.

- El VMM es capaz de modificar partes de la arquitectura de acogida que son difíciles de virtualizar. Esto requiere cambios en el sistema operativo invitado, pero no software de usuario.

Para llevar a cabo estas funciones, el VMM ocupa una posición privilegiada en el sistema a partir de arquitectura basada en capas discutida anteriormente, cuando se hablaba de paravirtualización.

El dominio privilegiado o conductor (dominio 0) es un sistema operativo especial privilegiado que realiza tareas administrativas para el hipervisor. Lanzado en el arranque, es la herramienta principal para manipular y migrar dominios invitados entre máquinas físicas. Dom0 crea conductores ideales para dispositivos de red y de bloques de todos los invitados y se comunica con los clientes mediante transporte asíncrono por memoria compartida.

El Dom0 también ejecuta un controlador en segundo plano que se encarga de proporcionar a cada invitado una interfaz genérica para crear la ilusión de que el dispositivo está dedicado a cada máquina virtual. En Dom0 hay dos herramientas para la gestión de las máquinas virtuales:

1) xend (*xen daemon*): Se trata de un proceso crítico de Xen que se ejecuta como *root* en Dom 0. Proporciona una interfaz denominada xm (*xen management*) a los usuarios finales con el fin de crear, cerrar o manipular dominios y su configuración de recursos. Tanto xend como xm son *scripts* de Python.

2) xenstored (*xen store daemon*): Xen proporciona una manera de compartir la información de configuración entre varios invitados para el mantenimiento de una base de datos común que se denomina XenStore. Se utiliza sobre todo para el control de dispositivos en dominios invitados, llevar a cabo operaciones atómicas como la lectura/escritura de las claves, y manipular el estado de configuración entre los conductores.

Los sistemas operativos modificados que se ejecutan sobre un VMM gestionado por Dom0 se denominan dominios invitados o de usuario (Dom U). Distintas aplicaciones de usuario se ejecutan en Dom U, como aquellas que interactúan con los controladores de dispositivos. A continuación veremos breve-

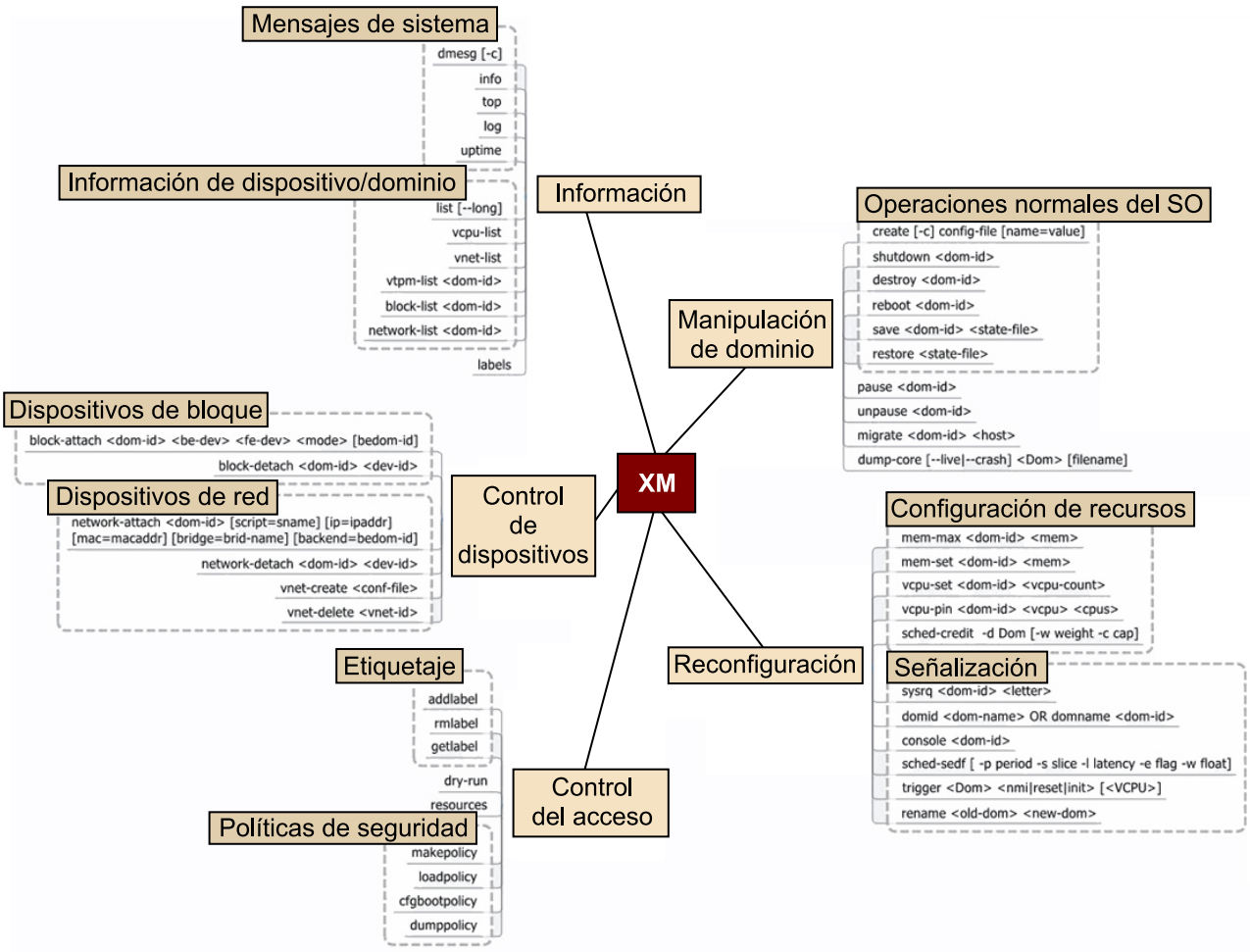
mente algunas herramientas de Xen que tienen una gran cantidad de funcionalidades dirigidas a controlar/monitorizar el comportamiento de estas máquinas virtuales invitadas y del VMM en general.

El comando xm se utiliza para la administración de los dominios invitados en Xen. Sus principales **funciones** (agrupadas por categorías en la figura 20) son:

- Gestión de los diferentes dispositivos.
- Manipulación de las máquinas virtuales invitadas y la reconfiguración de diferentes recursos, como CPU, memoria, etc.
- Diseño de políticas.

La información del sistema también se puede consultar por motivos de depuración o de control.

Figura 20. Funciones del comando xm de Xen



Xen proporciona una herramienta denominada XenTop, parecida al *top* característico de los sistemas Linux, que se puede ejecutar como programa interactivo en todas las máquinas virtuales en ejecución y que utiliza un formato de tabla con estadísticas de consumo de recursos. Es una herramienta fácil de usar

para comprender rápidamente el estado del sistema y para observar el efecto de ciertas acciones. La lista de la información proporcionada es dinámica y puede incluir el porcentaje de uso de CPU y memoria, tráfico de red, etc.

También existe la herramienta XenMon, que se encarga de monitorizar la calidad de servicio proporcionada y en general del rendimiento. Basado en esta monitorización, también permite realizar ciertas acciones para cumplir con los objetivos de las políticas establecidas.

3.7. Computación *cloud* para altas prestaciones

Tal como hemos visto, la computación *cloud* está revolucionando el mundo de la empresa de la misma manera que Internet lo hizo no hace tanto tiempo. La computación *cloud* está cambiando aspectos fundamentales en el modo como las empresas gestionan la infraestructura de las tecnologías de la información, tanto interna como externamente, proporcionando acceso bajo demanda a los servicios de computación con la capacidad de escalar los recursos cuando es necesario mediante la externalización de infraestructura.

Al mismo tiempo que la computación *cloud* está redefiniendo las tecnologías de la información, la creciente escala de computación y volúmenes de datos está cambiando la manera de hacer ciencia e investigación en ingeniería, con nuevos paradigmas y prácticas, que se basan esencialmente en los datos y la colaboración para afrontar grandes retos a escala internacional. Se espera que los servicios *cloud* se puedan unir a los sistemas clúster tradicionales de altas prestaciones y sistemas de cómputo *grid* para realizar descubrimientos científicos clave.

Los *clouds* pueden permitir la externalización/subcontratación de muchos aspectos de la investigación y de la educación que no son realmente útiles, como la implementación, configuración y administración de la infraestructura, y así permitir a los científicos centrarse en la ciencia. Los servicios *cloud* también pueden mejorar la productividad, facilitar el intercambio de resultados de investigación y permitir la reproductibilidad de los cálculos asociados. Por otro lado, más similar al entorno empresarial, los *clouds* pueden democratizar el acceso a los recursos computacionales y de datos (mediante el acceso a los investigadores que no tienen la infraestructura local adecuada), lo que ha demostrado un impacto significativo en la productividad de la investigación (IBM, 2009). De hecho, una reciente encuesta de usuarios del Departamento de Energía de Estados Unidos llevada a cabo por el equipo Magellan observó que las motivaciones principales para el movimiento de los usuarios hacia el *cloud* era la facilidad de acceso a los recursos informáticos (citado por el 79%), la capacidad de controlar los entornos de software (59%) y la capacidad de compartir la configuración de software y los experimentos con los compañeros (52%).

Sin embargo, también es importante mirar más allá de los beneficios de la externalización y comprender las posibles maneras de formular las aplicaciones y de utilizar los recursos en un entorno híbrido en el que se pueden combinar sistemas de altas prestaciones tradicionales, sistemas de computación *grid* y sitemas de computación *cloud*.

La computación *cloud* se está sumando a los sistemas de altas prestaciones tipo clúster y en *grid* como plataformas viables para aplicaciones de ciencia e ingeniería. Así, es especialmente importante descubrir nuevas formas de uso de toda esta infraestructura para dar soporte a aplicaciones de la ciencia e ingeniería. Para llevarlo a cabo, es importante entender bien las aplicaciones y sus requisitos. Por ejemplo, habrá que tener claro si se trata de aplicaciones de altas prestaciones de computación (HPC), de *throughput* (HTC) o masiva de tareas (MTC) (Raicu y otros, 2008). Las aplicaciones HPC están bien acopladas a grandes cantidades de comunicación interprocesador, y normalmente requieren grandes cantidades de potencia de cálculo durante periodos de tiempo cortos. Por otro lado, las aplicaciones de HTC están en general poco acopladas y en ellas la comunicación entre procesadores es limitada o inexistente. Las aplicaciones HTC también requieren grandes cantidades de computación, pero para plazos mucho más largos (meses o años, en lugar de horas o días). Finalmente, las aplicaciones de MTC son un híbrido de las dos clases anteriores. Las aplicaciones MTC pueden consistir en tareas de acoplamiento flexible, donde cada una de estas tareas es una aplicación muy acoplada que se ejecuta durante un periodo corto de tiempo (es decir, segundos o minutos).

Los servicios de computación *cloud* pueden soportar aplicaciones científicas de múltiples formas. Pueden proporcionar una plataforma para las aplicaciones, por ejemplo, cuando la infraestructura local no está disponible. También pueden servir para complementar plataformas existentes para proporcionar capacidad adicional o capacidades complementarias para satisfacer las necesidades heterogéneas y dinámicas. Por ejemplo, los *clouds* pueden servir como aceleradores, o proporcionar más resiliencia por el hecho de poder disponer de recursos adicionales en el supuesto de que se produzca un error. De hecho, la computación *cloud* no solo puede ayudar a los científicos a resolver los problemas de hoy con más eficacia, sino que también puede permitir explorar nuevas maneras de formular los problemas mediante las abstracciones de la computación *cloud*, como son la elasticidad o el modelo de pago por uso.

Hay que tener en cuenta que nos centramos en los *clouds* de computación pero la misma discusión también se puede aplicar a los *clouds* de datos; la idea es poder soportar datos científicos y de ingeniería y proporcionar el análisis de datos como servicio.

Entre los posibles modelos que combinan la computación *cloud* con los conceptos más tradicionales de la computación de altas prestaciones, podemos encontrar:

1) HPC *cloud*, donde los investigadores pueden externalizar aplicaciones completas en el *cloud* y/o plataformas de *cloud* privado. Los sistemas *cloud* actuales pueden resultar eficaces para ciertas clases de aplicaciones, como las HTC. Un reciente informe técnico (Fox y Gannon, 2012) ha estudiado exhaustivamente cómo ejecutar aplicaciones HPC *cloud*. Según este estudio, la computación *cloud* solo funciona con eficacia con ciertos tipos de aplicaciones HPC. Un ejemplo de esto son las aplicaciones *embarrassingly parallel*, que analizan datos independientes o generan simulaciones independientes que integran datos de sensores distribuidos, o de análisis de datos que pueden utilizar modelos de tipo MapReduce. Otros trabajos de investigación (Fox, 2011; Iosup y otros, 2011) muestran que las diferentes variantes de cálculos de MapReduce y MapReduce iterativo funcionan bien en *cloud*. En general, las aplicaciones HPC con sincronización mínima y requisitos de comunicaciones mínimas, con poca entrada/salida y de escalas modestas, son adecuadas para las plataformas de *cloud* actuales y, por lo tanto, pueden ser externalizadas en el *cloud* con éxito. El coste y el rendimiento relativo pueden resultar un problema. Por ejemplo, el informe del proyecto Magellan (Yelick y otros, 2011) señala que los servicios *cloud* resultaron de 7 a 13 veces más caros.

2) HPC más *cloud*, se centra en la exploración de escenarios en los que los *clouds* pueden complementar recursos HPC / *grid* con los servicios *cloud* para dar soporte a aplicaciones de ciencia y de ingeniería, por ejemplo, y soportar así requisitos heterogéneos, picos inesperados en la demanda, etc. Un *cloud* híbrido más infraestructura HPC tradicional también pueden, potencialmente, permitir nuevas formulaciones para aplicaciones que utilizan *clouds* como aceleradores, por resiliencia, o para poder balancear el compromiso entre coste/consumo eléctrico/rendimiento.

3) HPC como un *cloud*, se centra en exponer recursos HPC / *grid* utilizando las abstracciones de *cloud*, con el objetivo de combinar la flexibilidad de los modelos del *cloud* con el rendimiento de los sistemas HPC. Debido a las limitaciones de los *clouds* para soportar aplicaciones HPC directamente, los proveedores del *cloud* se dieron cuenta de la necesidad de proporcionar soluciones *cloud*, que se construyen específicamente para aplicaciones HPC (es decir, el hardware con procesadores e interconexiones más rápidas). Algunos proveedores han proporcionado incluso hardware no virtualizado para proporcionar el rendimiento que estas aplicaciones necesitan. Esto se conoce como HPC como un *cloud* (es decir, ejecutar aplicaciones HPC en los recursos que se exponen como recursos bajo demanda utilizando abstracciones del *cloud* para aprovechar el modelo de *cloud* sin sacrificar el rendimiento HPC que las aplicaciones científicas requieren). Existen dos enfoques principales que se utilizan para proporcionar HPC como un *cloud*: el primero utiliza grandes sistemas HPC que se pueden aprovisionar a modo de *clouds*; el segundo utiliza pequeños clústeres HPC que se pueden conectar entre sí para formar un gran *cloud*. Estos clústeres HPC pueden ser virtualizados o no virtualizados para ofrecer un mejor rendimiento.

Bibliografía

Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; Zaharia, M. (2010). "A view of cloud computing". *Commun. ACM* (núm. 53, vol. 4, pág. 50-58).

Foster, I.; Kesselman, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Burlington, Massachusetts: Morgan Kaufmann.

Foster, I.; Kesselman, C.; Tuecke, S. (2001). "The Anatomy of the Grid, Enabling Scalable Virtual Organizations". *International Journal of High Performance Computing Applications* (vol. 15, núm. 3).

Foster, I.; Yong Zhao; Raicu, I.; Lu, S. (2008). "Cloud Computing and Grid Computing 360-Degree Compared. Grid Computing Environments Workshop, 2008". *GCE '08* (pág. 1-10 y 12-16).

Fox, G. (2011). "Data Intensive Applications on Clouds". *Keynote at The Second International Workshop on Data Intensive Computing in the Clouds (DataCloud-SC11) at SC11 November 14*.

Fox, G.; Gannon, D. (2012). "Cloud Programming Paradigms for Technical Computing Applications" (Informe técnico).

Goyal, A.; Dadizadeh, S. (2009). "A Survey on Cloud Computing". Columbia Británica: The University of British Columbia.

Hwang, K,; Dongarra, J.; Fox, G. (2011). *Distributed and Cloud Computing*. Burlington, Massachusetts: Morgan Kaufmann.

IBM (2009). "IBM Research Doubles Its Productivity with Cloud Computing". *Case Study QuickView*.

Iosup, A.; Ostermann, S.; Yigitbasi, N.; Prodan, R.; Fahringer, T.; Epema, D. (2011). "Performance analysis of Cloud computing services for many-tasks scientific computing". *IEEE TPDS*. Los Alamitos, CA.

Iverson, W. (2004). *Real World Web Services*. Sebastopol, CA: O'Reilly (recurso electrónico).

Mell, P.; Grance, T. (2011). *The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology*. U.S. Department of Commerce.

Monson-Haefe, R. (2004). *J2EE Web services*. Reading, Massachusetts: Addison-Wesley.

Raicu, I.; Zhang, Z.; Wilde, M.; Foster, I.; Beckman, P.; Iskra, K.; Clifford, B. (2008). "Towards Loosely-Coupled Programming on Petascale Systems". *IEEE/ACM Supercomputing*.

Yelick, K. y otros (2011). *The Magellan Report on Cloud Computing for Science*. U.S. Department of Energy Office of Advanced Scientific Computing Research (ASCR).