

**Tecnológico Nacional de México**

**Instituto Tecnológico de Reynosa**

Materia: Sistemas Operativos 2

ACTIVIDAD13: Investigación exhaustiva del Tema06: Seguridad con sus 4 subtemas.

Tema 6: Usos y tendencias de los sistemas distribuidos

Alumno: Castillo Jr. Gregorio

Numero de control: 19580589

Correo electrónico: L19580589@reynosa.tecnm.mx

7mo Semestre Matutino Salón 7

Docente: Mario José Santiago Sánchez

Fecha de entrega: 06/12/2022



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Consecutivo** | **Numero de**  **control** | **Apellido Paterno** | **Apellido Materno** | **Nombres(s)** | **Correo electrónico**  **Institucional** | **Firma de que está**  **autorizando que se**  **entregue esta Publicación** | **Fotografía del rostro de cada**  **Integrante del equipo (selfie)** |
| **1** | **19580589** | **Castillo Jr** |  | **Gregorio** | **L@19580589@reynosa.tecnm.mx** |  |  |
| **2** | **19580595** | **Flores** | **Acosta** | **Sheila Lizeth** | **L@19580595@reynosa.tecnm.mx** |  |  |
| **3** | **19580867** | **Morales** | **Calixto** | **Daniel Alexander** | **L@19580867@reynosa.tecnm.mx** | No hay descripción disponible. |  |

Equipo #12

# Tabla de contenidos

[Tabla de contenidos 3](#_Toc121170804)

[Contenido 3](#_Toc121170805)

[**6.1 Sistemas basados en objetos distribuidos** 4](#_Toc121170806)

[**6.2 Sistemas de archivos distribuidos** 11](#_Toc121170807)

[**6.3 Sistemas distribuidos basados en la Web** 18](#_Toc121170808)

[**6.4 Sistemas distribuidos basados en coordinación** 24](#_Toc121170809)

[**Conclusiones** 28](#_Toc121170810)

[**Herramientas y recursos** 29](#_Toc121170811)

[**Bibliografía** 30](#_Toc121170812)

# Contenido

## **6.1 Sistemas basados en objetos distribuidos**

Debemos de saber que un objeto distribuido dentro de los sistemas cliente/servidor es aquel que es gestionado por un servidor y los clientes invocan sus métodos utilizando un método de invocación remota. El cliente puede invocar el método mediante un mensaje al servidor que gestiona el objeto, se ejecuta el método por el objeto en el servidor y el resultado se le devuelve al cliente en otro mensaje.

**ARQUITECTURA**

Dentro de la orientación a objetos hay un importante paradigma en el desarrollo de software. Desde que fue la presentación de esto, ha tenido una gran popularidad. La popularidad se debió a la derivación de la habilidad natural de construir software en la forma de componentes independientes. Los desarrolladores pueden concentrarse en implementar una sola funcionalidad especifica independiente de los desarrolladores.

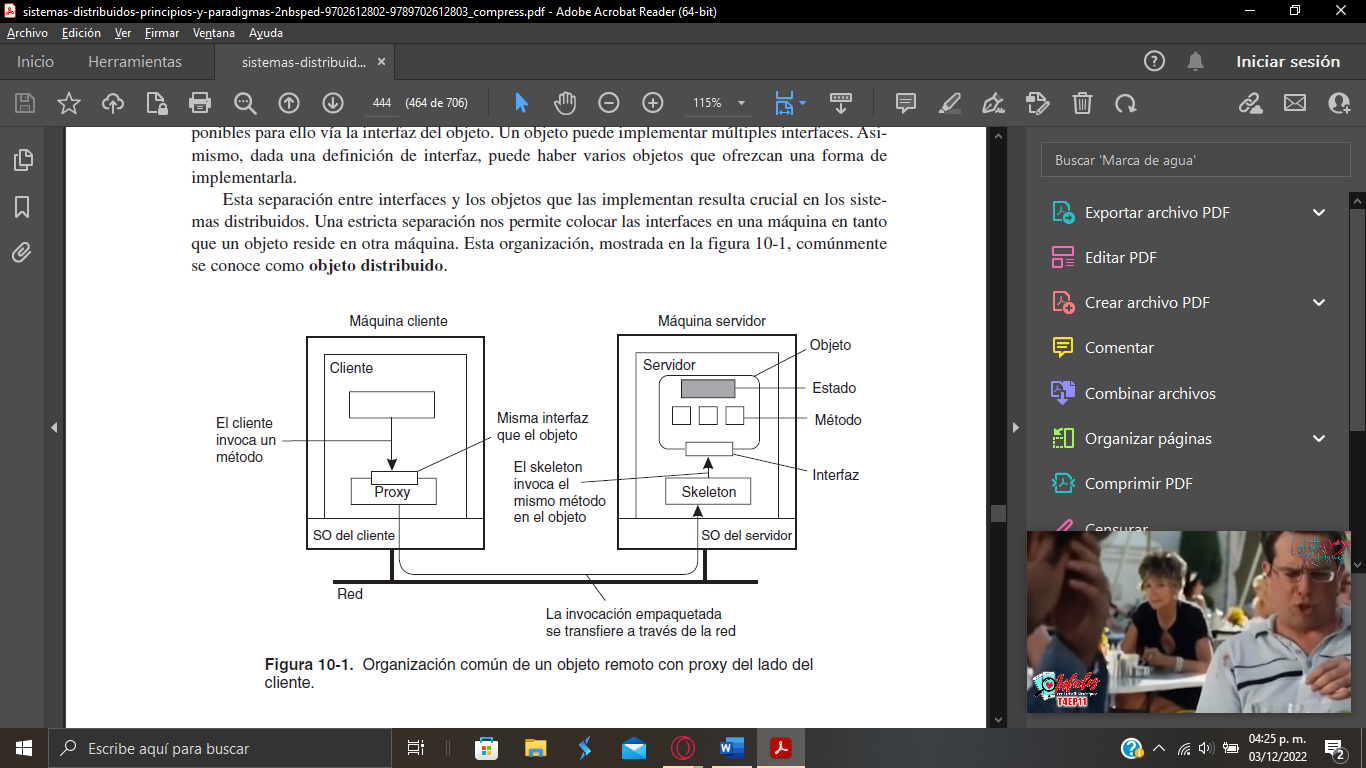
Desde 1980 la orientación de objetos fue utilizada para desarrollar sistemas distribuidos. La noción de algún objeto independiente alojado en un servidor remoto al mismo tiempo que se podría lograr la transparencia de distribución formo una base solida para el desarrollo a una nueva generación.

**Objetos distribuidos**

Recapitulando sobre los objetos distribuidos nos encontramos en una posición donde encontramos que una de las características fundamentales de los objetos es que encapsula los datos, a esto se le llama el *estado*, y a las operaciones incluidas en tales datos se les llama *métodos*, y a estos métodos se ponen a disposición mediante una *interfaz*.

Aquí lo importante seria entender que no existe alguna forma “legal” de que un proceso pueda acceder o manipular el estado de un objeto que no sea a través de la invocación de los métodos disponibles y para eso la vía interfaz del objeto. Sabemos que un objeto puede implementar varias interfaces. Así igual en una interfaz se puede haber varios objetos que ofrezcan distintas maneras de implementar.

La separación de las interfaces y los objetos que han sido implementados siempre resultan de manera crucial en los sistemas distribuidos. Y sí pasa una estricta separación permite colocar las interfaces en una maquina mientras que un objeto reside en otra máquina. Este tipo de organización es conocida como objeto distribuido.



Cuando el cliente se vincula al objeto distribuido, una implementación de interfaz de objeto a la cual se le llama proxy, se encarga del espacio del espacio destinado a la dirección del cliente.

El proxy es análogo a una plantilla del cliente en sistemas RPC. Lo único que hace es organizar las invocaciones a métodos en mensajes y desorganizar los mensajes de la respuesta para poder revolver el resultado de la invocación al método del cliente. El objeto real esta en la en la maquina de un servidor, donde nos ofrece la misma interfaz que en la maquina del cliente. Las solicitudes de las invocaciones entrantes siempre se pasan resguardadas en el servidor, el cual esta desempaquetado para poder crear invocaciones a métodos en la interfaz del objeto ubicada en el servidor. El resguardado que esta en el servidor también es responsable de desempaquetar las respuestas y poder remitir mensajes al proxy del lado del cliente.

El resguardo se le conoce como plantilla de código el cual en términos técnicos se le llama *skeleton*, ya que lo proporciona los medios básicos para permitir que el middleware del servidor acceda a los objetos que fueron definidos por el usuario. Ya aplicado en casos reales, frecuentemente contiene un código incompleto en la forma de una clase propia de un lenguaje que tiene especializado por el desarrollador.

**Tiempo de compilación versus objetos en tiempos de compilación**

En sistemas distribuidos los objetos suelen aparecer en varias formas. La que resulta más evidente es la directamente relacionada con el objeto a nivel lenguaje, tales como aquellos que son soportados por Java, C++ u otros lenguajes orientados a objetos y son conocidos como objetos en tiempo de compilación.

La utilización de los objetos en tiempo de compilación dentro de los sistemas distribuidos facilita la construcción de aplicaciones distribuidas. La desventaja es que los objetos en tiempo de compilación dependen de un lenguaje de programación.

Cuando se habla de objetos en tiempo, la manera de como se incrementan es dejándolo abierto. Es decir, un desarrollador puede decidir escribir una biblioteca C que contenga varias funciones que trabajen en un archivo de datos en común.

**Objetos persistentes y transitorios**

Un objeto persistente es uno que continúa existiendo aun cuando ya no este contenido en el espacio de dirección de cualquier proceso de servidor, es decir, que no depende de su servidor.

Un objeto transitorio es un objeto que solo existe mientras el servidor donde se aloja exista. Desde el momento en que el servidor deje de funcionar, el objeto deja de existir.

**PROCESOS**

Los servidores siempre juegan un papel muy importante en los sistemas distribuidos, en sí el servidor es diseñado para alojar objetos distribuidos.

**Servidores de objetos**

Es un servidor diseñado para poder soportar objetos distribuidos. La diferencia que hay entre un servidor general y otros servidores tradicionales es que un servidor de objetos no proporciona por sí mismo un servicio específico. Estos servidores específicos son implementados por los objetos que están residiendo en el servidor. Digamos que el servidor solo nos proporciona los medios necesarios para invocar objetos locales basados en solicitudes de clientes remotos.

El servidor de objetos es como un lugar donde residen los objetos. Un objeto se compone de dos partes:

* Los datos que representan su estado.
* El código para poder ejecutar los métodos.

**Alternativas para invocar objetos**

Para poder invocar un objeto el servidor necesitaría saber que código va ejecutar, en que datos va operar, si tendrá que iniciar un hilo distinto para que se haga encargo de la invocación y así sucesivamente. Un método simple es asumiendo que todos los objetos son iguales y que solo existe una forma de invocarlos, pero este método es considerado inflexible.

El mejor método para esto es que un servidor pueda soportar las diferentes políticas, por ejemplo, los objetos transitorios que son objetos que solo existen mientras el servidor siga existiendo, pero posiblemente en un lapso corto. Una copia de un archivo solo lectura guardada en la memoria que puede ser implementada como un objeto transitorio.

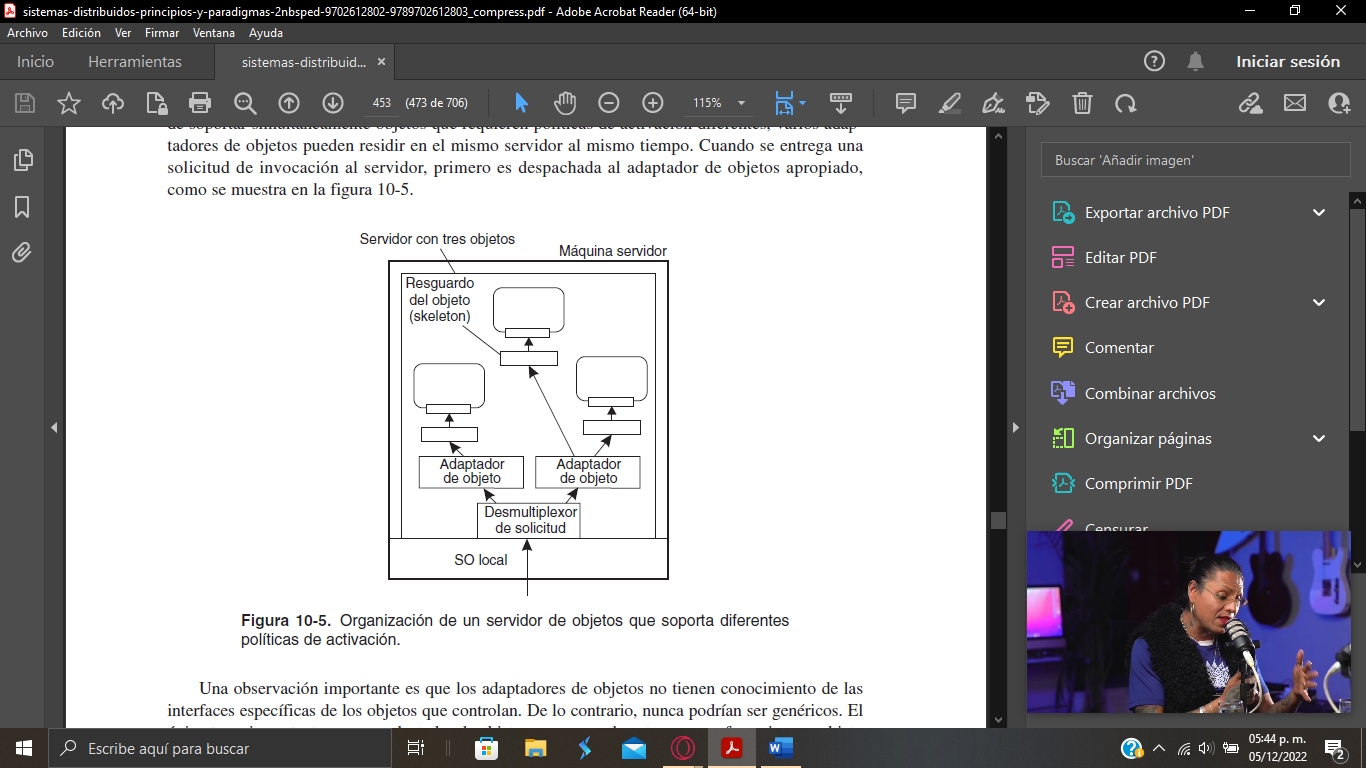
La ventaja que se obtiene de este método es que un objeto transitorio necesitara los recursos de un servidor solo en tato se requiera el objeto. Una desventaja seria que la invocación de estos puede llevarse mucho tiempo para que se pueda completar, ya que primero se tiene que crear el objeto. La política alternativa es poder crear objetos transitorios en el momento en que el servidor se inicia, a expensas del recurso incluso cuando ningún cliente utilice el objeto.

El método alternativo es permitirle que a los objetos por lo menos compartan el código. Ejemplo una base de datos contiene objetos que pertenecen a la misma clase, se pueden implementar eficientemente si la implementación de clase es cargada una vez en el servidor, y cuando llega una solicitud de invocación de un objeto, solo el servidor necesitaría buscar el estado del objeto en base de datos y ejecutar el método solicitado. Un beneficio de este tipo de método es que los objetos se quedan protegidos automáticamente contra el acceso concurrente.

**Adaptador de objetos**

Las decisiones de invocar objetos comúnmente son conocidas como política de activación, para poder enfatizar en muchos de los casos el propio objeto primero tiene que ser llevado al espacio dirección del servidor activado antes de que en realidad puede ser invocado. Lo que se requiere un mecanismo para que se pueda agrupar objetos por política, este mecanismo se denomina adaptador de objetos o envolvedor de objetos. Un adaptador puede considerarse mejor como un software que implementa una política de activación física.

El adaptador de objetos puede controlar uno o mas objetos. Debido que un servidor debe de ser capaz de soportar simultáneamente objetos que requieren de políticas de activación distintas, varios adaptadores de objetos pueden residir en el mismo servidor al mismo tiempo.



**COMUNICACIÓN**

Este mecanismo se basa en gran medida en las llamadas a los procedimientos remotos los cuales se abordan extensamente.

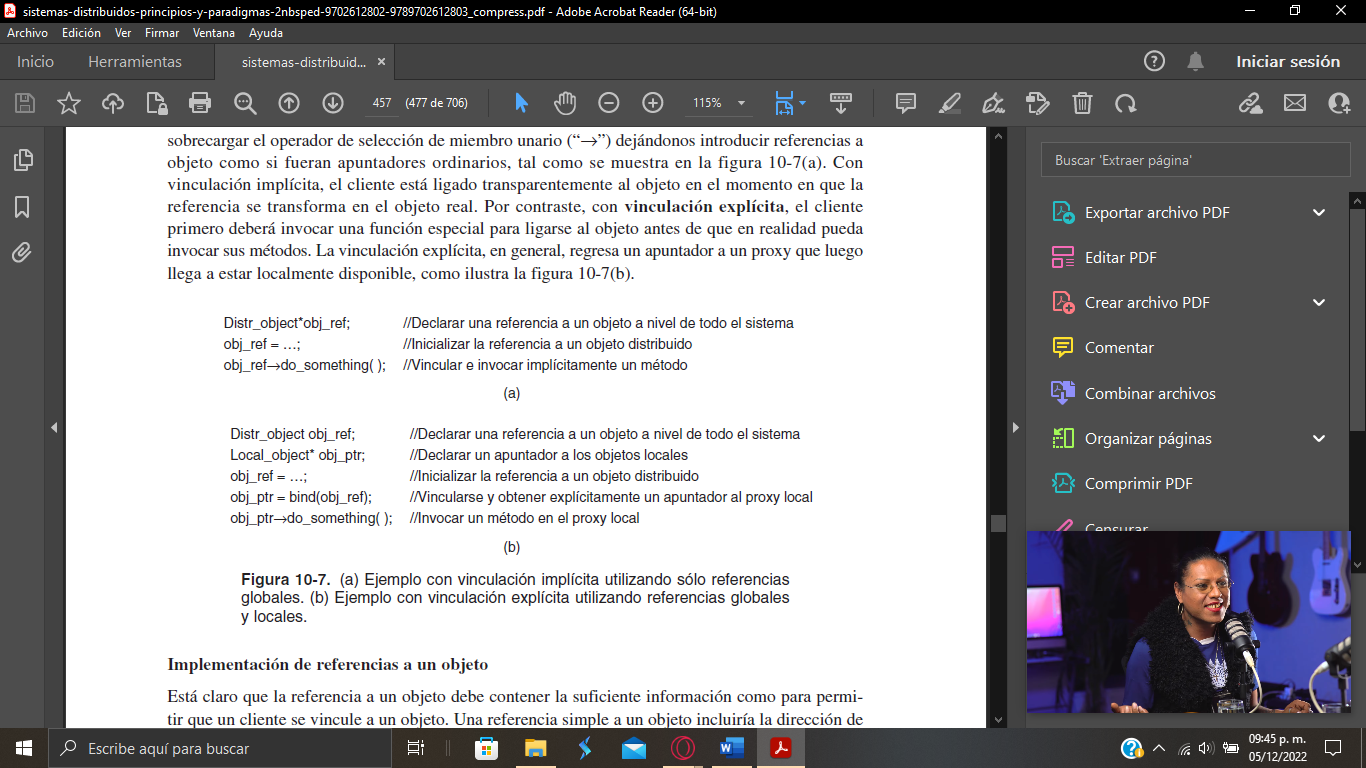
**Vinculación de un cliente a un objeto**

Una diferencia entre los sistemas RPC tradicionales que soportan los objetos distribuidos es que estos, en general, proporciona referencias a objetos a nivel de todo el sistema. Esas referencias a objetos pueden ser pasadas libremente entre los procesos en distintas maquinas, por ejemplo, con parámetros ante invocaciones a métodos.

Cuando un proceso retiene referentemente un objeto, primero debe de vincular el objeto referido antes de invocar cualquier de los métodos. La vinculación hace que el proxy en el espacio de dirección del proceso, así se puede implementar una interfaz que contiene los métodos que el proceso invoca. En varios casos la invocación puede ser automáticamente.

Con vinculación implícita el cliente se le ofrece un mecanismo simple que le permite invocar directamente los métodos que utilizan una referencia a un objeto, de ejemplo tenemos a C++ que nos permite sobrecargar el operador de selección de miembro unario dejando introducir referencias a los objetos como si fuesen apuntadores ordinarios. La vinculación implícita el cliente se liga transparentemente en el objeto en el momento en que la referencia se transforma en el objeto real.

En contraste, la vinculación explicita el cliente primero deberá invocar una función especial para poder ligarse al objeto antes de la realidad pueda invocar los métodos, en sí, regresa un apuntador a un proxy que luego llega a estar de manera local disponible.



**Implementación de referencias a un objeto**

La referencia a un objeto debe contener la suficiente información como para poder permitir que un cliente se vincule a un objeto. Una referencia simple a un objeto incluiría la dirección de red de la maquina donde está residiendo el objeto, y junto a esto un punto final que identifica el servidor que maneja el objeto, más una indicación acerca de que objeto se trata.

Si la máquina del servidor se congela y el servidor se le asigna un punto de terminación diferente después de recuperarse, todas las referencias a un objeto se vuelven invalidas. El problema se resuelve con hacer centinela local por cada maquina escuche a punto de terminación bien conocido y no se pierda de vista las asignaciones del servidor punto de terminación en una tablita de puntos de terminación.

## **6.2 Sistemas de archivos distribuidos**

Un sistema de archivos distribuido, o DFS, lo podemos ver como un esquema de almacenamiento y de gestión de datos que nos permite a nosotros los usuarios o las aplicaciones acceder a archivos de datos como PDF, documentos de Word como el que estamos escribiendo en este momento, archivos de video, archivos de audio, etc., desde un almacenamiento compartido en cualquiera de los múltiples servidores de red. Sus datos compartidos y almacenados en un clúster de servidores permiten a muchos de usuarios compartir recursos de almacenamiento y archivos de datos en múltiples equipos.

Como sabemos hay dos razones principales por las que una empresa usaría un DFS:

* Para almacenar datos de forma permanente en soportes de almacenamiento secundario
* Para compartir información de forma fácil, eficiente y segura entre usuarios y aplicaciones

Como subsistema del sistema operativo del equipo, mediante DFS se gestiona, organiza, almacena, protege, recupera y comparte los archivos de datos. Las aplicaciones o los usuarios pueden almacenar o acceder a los archivos de datos en el sistema como lo harían con un archivo local. Desde sus ordenadores o teléfonos inteligentes, los usuarios pueden ver todas las carpetas compartidas de DFS como una ruta única que se ramifica en una estructura arbolada a los archivos almacenados en varios servidores.

El DFS tiene dos componentes críticos:

* **Transparencia de la ubicación –** esto significa que los usuarios verán un único espacio de nombres para todos los archivos de datos, independientemente del ordenador que utilicen para acceder o almacenar los archivos. Los usuarios no podrán saber dónde se almaceno el archivo por primera vez y podrán mover archivos dentro de las carpetas según sea necesario sin tener que cambiar el nombre de la ruta.
* **Redundancia –** mediante una característica de replicación de archivos, DFS extiende copias de un archivo a través de los nodos del clúster, lo que significa que los datos permanecen altamente disponibles, incluso en caso de fallo del servidor.

Una técnica muy conocida es desplegar técnicas de distribución de archivos, mediante las cuales un archivo se distribuye a través de múltiples servidores. La idea básica es simple: distribuyendo un archivo grande entre múltiples servidores, es posible buscar sus diferentes partes en paralelo. Desde luego, semejante organización funciona bien sólo si la aplicación se organiza de tal forma que el acceso a los datos en paralelo tenga sentido.

¿Cómo funcionan los sistemas de archivos distribuidos?

Mediante DFS, los terminales y servidores se conectan en red para crear un sistema de archivos paralelo con un clúster de nodos de almacenamiento. El sistema de agrupa bajo un único espacio de nombres y un grupo de almacenamiento y puede permitir el acceso rápido a los datos a través de varios hosts, o servidores, simultáneamente.

Los datos en si pueden residir en diversos dispositivos o sistemas de almacenamiento, desde unidades de disco duro (HDD) hasta unidades de estado sólido (SSD) y la cloud pública. Independientemente de donde se almacenan los datos, DFS se puede configurar como un espacio de nombres autónomo (o independiente), con solo un servidor host o un espacio de nombres basado en dominios con varios servidores host.

Pero cuando un usuario hace un clic en un nombre de archivo para acceder a esos datos, el DFS comprueba varios servidores, dependiendo de donde se encuentre al usuario, y luego sirve la primera copia disponible del archivo en ese grupo de servidores. Esto es lo que evita que cualquiera de los servidores se atasque demasiado cuando muchos usuarios acceden a los archivos y también mantiene los datos disponibles a pesar de que el servidor funcione mal o falle.

A través de esta función de replicación de archivos DFS, cualquier cambio realizado en un archivo se copia en todas las instancias de ese archivo en los nodos del servidor.

**Características de los sistemas de archivos distribuidos**

Hay muchas soluciones DFS diseñadas para ayudar a las empresas, a gestionar, organizar y acceder a sus archivos de datos, pero la mayoría de esas soluciones incluyen diferentes características:

* **Transparencia de acceso –** los usuarios acceden a los archivos como si estuvieran almacenados localmente en sus propios terminales.
* **Transparencia de la ubicación –** las maquinas host no necesitan saber dónde se encuentran los datos del archivo porque el DFS lo gestiona
* **Bloqueo de archivos –** el sistema bloquea los archivos en uso en todas las ubicaciones para evitar que dos usuarios de diferentes ubicaciones hagan cambios en el mismo archivo del mismo tiempo
* **Cifrado de datos en tránsito –** DFS protege los datos cifrados a medida que se mueven por el sistema,
* **Compatibilidad con varios protocolos –** los hosts pueden acceder a los archivos mediante una variedad de protocolos, como Server Message Block (SMB), Network File System (NFS) y Portable Operating System Interface (POSIX).

¿Cuáles son las ventajas de los sistemas de archivos distribuido?

Una de las principales ventajas de un sistema de archivos distribuido es que permite acceder a los mismos datos desde muchos lugares. También hace que el intercambio de información en todas las geografías sea simple y extremadamente eficiente. DFS puede eliminar por completo la necesidad de copiar archivos de un sitio a otro o de mover carpeta, todo lo cual requiere tiempo y esfuerzo que se pueden invertir mejor en otro lugar.

Otras de las ventajas más destacadas y beneficios son:

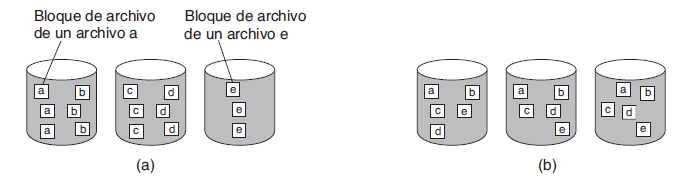
* **Resiliencia de los datos –** como los archivos residen en más de una ubicación, un fallo del servidor no será señal de desastre.
* **Eficiencia de la red –** las cargas de trabajo pesadas no ralentizaran el sistema porque DFS puede recopilar datos del siguiente nodo disponible.
* **Acceso a la información más reciente –** los cambios realizados en las carpetas o archivos compartidos son visibles y están disponibles al instante para todos los que utilizan el DFS.
* **Escalabilidad sencilla –** para hacer crecer el sistema basta con añadir más nodos. Alta fiabilidad la perdida de datos es mucho menos preocupante cuando los archivos se replican entre hosts.

Sistemas de archivos distribuido basados en clúster

El NFS es un ejemplo típico de muchos sistemas de archivo distribuidos, los cuales generalmente se organiza con arreglo a una arquitectura – servidor tradicional. Esta arquitectura a menudo se mejora agrupando los servidores con pocas diferencias.

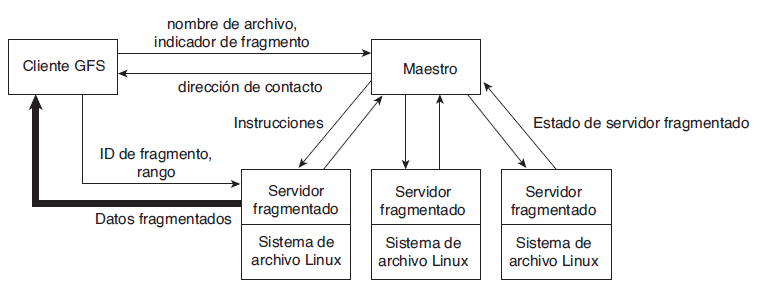
Pero si lo consideramos que los grupos de servidores a menudo se utilizan e aplicaciones en paralelo, no nos sorprende que en los sistemas de archivos asociados se ajusten como corresponde. Una técnica muy conocida es desplegar técnicas de distribución de archivos, mediante las cuales un archivo se distribuye a través de múltiples servidores. La idea básica es simple: distribuyendo un archivo grande entre múltiples servidores, es posible buscar diferentes partes en paralelo. Desde luego, ya está semejante organización funciona bien solo si la aplicación se organiza de tal forma que el acceso a los datos, tal como se guardan en el archivo tengan una estructura muy regular, por ejemplo, una matriz (densa).

Para las aplicaciones de propósito general, o para aquellas con tipos irregulares o muchos tipos de estructuras de datos, la distribución de archivos puede no ser una herramienta efectiva. En esos casos, a menudo es un poco más conveniente particional el sistema de archivo en su totalidad y simplemente guardar los diferentes archivos en diferentes servidores, pero no particional un solo archivo a través de múltiples servidores.



Más interesantes son los casos en que se debe organizar un sistema de archivos distribuidos para grandes centros de datos, tales como los utilizados por compañías como Amazon y Google. Estas compañías ofrecen servicios a clientes web cuyo resultado son lecturas y actualizaciones de un número masivo de archivos distribuidos a través de literalmente decenas de miles de computadoras. En tales entornos, las suposiciones tradicionales en relación con sistemas de archivo distribuidos ya no son válidas. Por ejemplo, es de esperarse que en cualquier momento falle una computadora.

Para abordar estos problemas, Google, por ejemplo, desarrolló su propio sistema de archivo Google (GFS, por sus siglas en inglés), cuyo diseño se describe en Ghemawat y colaboradores (2003). Los archivos Google tienden a ser muy grandes, comúnmente de varios gigabytes, y cada uno contiene muchos objetos más pequeños. Además, las actualizaciones de los archivos generalmente se realizan anexando datos en lugar de sobrescribir algunas partes de un archivo. Estas observaciones, junto con el hecho de que las fallas de servidor son la norma en lugar de la excepción, llevaron a la construcción de grupos de servidores



Cada grupo de GFS se compone de un servidor maestro junto con múltiples servidores fragmentados. Cada archivo GFS es dividido en fragmentos de 64 Mbytes cada uno, tras de lo cual estos fragmentos se distribuyen a través de los servidores llamados fragmentados. Una observación importante es que un GFS maestro se contacta sólo para recabar información de metadatos. En particular, un cliente GFS transfiere un nombre de archivo y un índice de fragmento al maestro, y espera una dirección de contacto para el fragmento. La dirección de contacto contiene toda la información necesaria para acceder al servidor fragmentado correcto y obtener el fragmento de archivo requerido.

Con esta finalidad, el GFS maestro esencialmente mantiene un espacio de nombre, adjunto a un mapeo desde un nombre de archivo hasta los fragmentos. Cada fragmento lleva un identificador asociado para que el servidor fragmentado pueda buscarla. Además, el servidor maestro no pierde de vista la ubicación de un fragmento. Los fragmentos se replican en prevención de fallas de manejador, pero nada más para eso. Una característica interesante es que el GFS maestro no intenta llevar la cuenta precisa de las ubicaciones de los fragmentos. En cambio, de vez en cuando se pone en contacto con los servidores fragmentados para ver qué fragmentos tienen guardados.

La ventaja de este esquema es la simplicidad. Observemos que el servidor maestro controla la asignación de los fragmentos a los servidores fragmentados. Además, éstos llevan la cuenta de lo que tienen guardado. Por consiguiente, una vez que el servidor maestro obtiene las ubicaciones de los fragmentos, tiene una imagen precisa de dónde están guardados los datos. Sin embargo, las cosas se complicarían si esta visión tuviera que ser consistente todo el tiempo. Por ejemplo, cada vez que un servidor fragmentado se congelara o cuando se agregara un servidor, el servidor maestro tendría que ser informado. En cambio, es mucho más simple refrescar su información a partir del conjunto actual de servidores fragmentados mediante encuestas. Los clientes GFS simplemente necesitan saber qué servidores fragmentados cree el servidor maestro que están guardando los datos solicitados. Como los fragmentos se replican de todos modos, existe una alta probabilidad de que un fragmento esté disponible al menos en uno de los servidores fragmentados.

¿Por qué se agranda este esquema? Un importante tema de diseño es que el servidor maestro está en control en gran medida, pero que no forma un cuello de botella a causa del trabajo que necesita realizar. Se han tomado dos tipos de medidas para manejar la escalabilidad.

La primera medida, y con mucho la más importante, es que la mayor parte del trabajo sea realizada por servidores fragmentados. Cuando un cliente necesita acceder a datos, se pone en contacto con el servidor maestro para indagar qué servidores contienen los datos. Después de eso, se comunica sólo con los servidores fragmentados. Los fragmentos se replican de acuerdo con un esquema de respaldo primario. Cuando el cliente está realizando una operación de actualización, se pone en contacto con el servidor fragmentado más cercano que contiene los datos y dirige sus actualizaciones a dicho servidor. Este servidor envía la actualización al siguiente más cercano que contiene los datos y así sucesivamente. Una vez que todas las actualizaciones se han propagado, el cliente se pondrá en contacto con el servidor fragmentado primario, el cual asignará entonces un número secuencial a la operación de actualización y la transferirá a los respaldos. Entre tanto, el servidor maestro se mantiene fuera del bucle.

La segunda medida, el nombre de espacio (jerárquico) para archivos, se implementa con una tabla de nivel único simple, en la cual los nombres de ruta se dirigen a metadatos (equivalentes a los nodos de los sistemas de archivo tradicionales). Además, toda la tabla se mantiene en la memoria principal, junto con la dirección de los archivos ubicados en las secciones. Las actualizaciones de estos datos se guardan en un almacenamiento persistente. Cuando el registro se vuelve demasiado grande, se utiliza un punto de control mediante el cual los datos que se encuentran en la memoria principal se guardan de tal forma que puedan ser tipografiados de vuelta en la memoria principal. Por consiguiente, la intensidad de E/S de un GFS maestro se reduce fuertemente.

Esta organización permite que un solo servidor maestro controle algunos cientos de servidores fragmentados, lo cual es una cantidad considerable para un solo grupo. Con la organización subsiguiente de un servicio tal como Google en servicios más pequeños tipografiados en grupos, no es difícil imaginar que se puede hacer que un enorme número de grupos funcionen juntos.

## **6.3 Sistemas distribuidos basados en la Web**

La Red mundial (WWW, del inglés World Wide Web) puede ser vista como un enorme sistema distribuido compuesto a partir de millones de clientes y servidores para poder acceder a documentos vinculados. Los servidores mantienen conjuntos de documentos, en tanto que los clientes proporcionan a los usuarios una interfaz de fácil uso para presentar y acceder a tales documentos.

La web se inició como un proyecto en CERN, el Laboratorio Europeo de Física de Partículas con sede en Ginebra, para permitir que su gran y geográficamente disperso grupo de investigadores tuviera acceso a documentos compartidos mediante un simple sistema de hipertexto. Un documento podría ser cualquier cosa que pudiera desplegarse en la terminal de un usuario en una computadora, tal como notas personales, informes, cifras, planos (blueprints), dibujos y así sucesivamente. Vinculando los documentos entre sí, fue fácil integrarlos desde diferentes proyectos en un nuevo documento sin la necesidad de realizar cambios centralizados. Lo único que se requería era construir un documento que proporcionara vínculos a otros documentos pertinentes

La web creció gradual y lentamente con sitios que no eran de física de alta energía, pero su popularidad se incrementó en forma impresionante cuando las interfaces de usuario gráficas estuvieron disponibles, principalmente Mosaic (Vetter y cols., 1994). Mosaic proporcionó una interfaz de fácil uso para presentar y acceder a documentos simplemente con hacer clic en un botón del ratón. Se buscaba un documento en un servidor, se transfería al cliente y se presentaba en la pantalla. Para un usuario, no había ninguna diferencia conceptual entre un documento guardado localmente o en otra parte del planeta. En este sentido, la distribución era transparente.

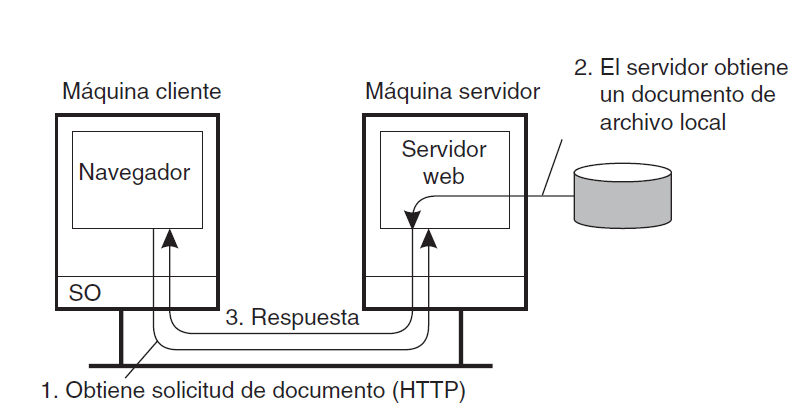
A partir de 1994, el World Wide Web Consortium (Consorcio de la red mundial) inició los desarrollos, una colaboración entre el CERN y el Massachusetts Institute of Technology. Este consorcio es responsable de estandarizar los protocolos, mejorar la interoperabilidad, y dar mayor realce a las capacidades de la web. Además, se puede advertir que por fuera de este consorcio se están dando muchos desarrollos nuevos que no siempre conducen a la compatibilidad deseada. Por ahora, la web no es más que un simple sistema basado en documentos. De manera notoria, con la introducción de servicios se está viendo el surgimiento de un enorme sistema de distribución donde se están utilizando, componiendo y ofreciendo servicios a cualquier usuario o máquina que sea capaz de utilizarlos.

Considerando que la web es muy reciente y que mucho ha cambiado en corto tiempo, nuestra descripción podría ser sólo una imagen instantánea de su estado actual. Sin embargo, como veremos, muchos conceptos que fundamentan la tecnología de la web.

A diferencia de muchos sistemas distribuidos presentados hasta ahora, los sistemas distribuidos basados en la web son relativamente nuevos. En este sentido, es un tanto difícil hablar sobre sistemas tradicionales basados en la web, aunque existe una clara distinción entre los sistemas que estuvieron disponibles al principio y los utilizados en la actualidad.

Muchos sistemas basados en la web están organizados como arquitecturas cliente-servidor relativamente simples. La parte central de un sitio web está conformada por un proceso que tiene acceso a un sistema de archivo local que guarda documentos. El modo más simple de referirse a un documento es por medio de una referencia llamada localizador uniforme de recursos (URL, por sus siglas en inglés). El URL especifica la localización de un documento, a menudo incluyendo un nombre DNS de su servidor asociado junto con el nombre del archivo mediante el cual el servidor puede buscar el documento en su sistema de archivo local. Además, un URL especifica el protocolo a nivel de aplicación para transferir el documento a través de la red. Existen varios protocolos diferentes disponibles.

Un cliente interactúa con servidores web a través de una aplicación especial conocida como navegador. Un navegador es responsable de desplegar apropiadamente un documento. También, un navegador acepta entradas de un usuario en su mayor parte para permitirle seleccionar una referencia a otro documento, el cual entonces el navegador busca y despliega. La comunicación entre un navegador y un servidor web se ha estandarizado: ambos se adhieren al protocolo de transferencia de hipertexto (HTTP, por sus siglas en inglés)



La web ha evolucionado en forma considerable desde su presentación. Por el momento, existe una inmensa variedad de métodos y herramientas que producen información que puede ser procesada por clientes web y servidores web. Sin embargo, pasamos por alto la mayoría de las herramientas y métodos utilizados para construir documentos web, pues con frecuencia no tienen una relación directa con la naturaleza distribuida de la web. Una buena introducción sobre cómo construir aplicaciones basadas en la web se encuentra en Sebesta (2006).

Documentos web

Para la web es fundamental que, virtualmente, toda la información llega en la forma de un documento. El concepto de un documento tiene que ser tomado en su sentido más amplio; no sólo puede contener texto simple, sino también incluir diversas clases de características dinámicas tales como audio, video y animaciones, y así sucesivamente. En muchos casos, se requieren aplicaciones de ayuda especiales para hacer que un documento “cobre vida”. Tales interpretadores típicamente estarán integrados al navegador de usuario.

La mayoría de los documentos pueden dividirse aproximadamente en dos partes: una parte principal que al final actúa como plantilla para la segunda parte, la cual se compone de muchos bits y piezas diferentes que en conjunto constituyen el documento desplegado en un navegador. La parte principal, en general, se escribe en lenguaje de marcas, muy similar al tipo de lenguajes utilizados en sistemas de proceso de palabras. El lenguaje de marcas que se utiliza más ampliamente en la web, el cual es acrónimo para lenguaje de marcas de hipertexto (HTML, por sus siglas en inglés). Como su nombre lo indica, el HTML permite insertar vínculos a otros documentos. Cuando se activan en un navegador, el documento referido será tomado de su servidor asociado.

Otro lenguaje de marcado cada vez más importante es el lenguaje de marcado extensible (XML, por sus siglas en inglés), el cual, como su nombre sugiere, es más flexible al definir cómo deberá verse un documento. La diferencia principal entre el HTML y el XML, es que éste incluye las definiciones de los elementos que marcan un documento. En otros términos, es un lenguaje de meta marcado. Este método permite una gran flexibilidad cuando se trata de especificar con exactitud cómo deberá verse un documento: no existe la necesidad de apegarse a un solo modelo en la forma que lo dicta un lenguaje de marcado fijo tal como el HTML.

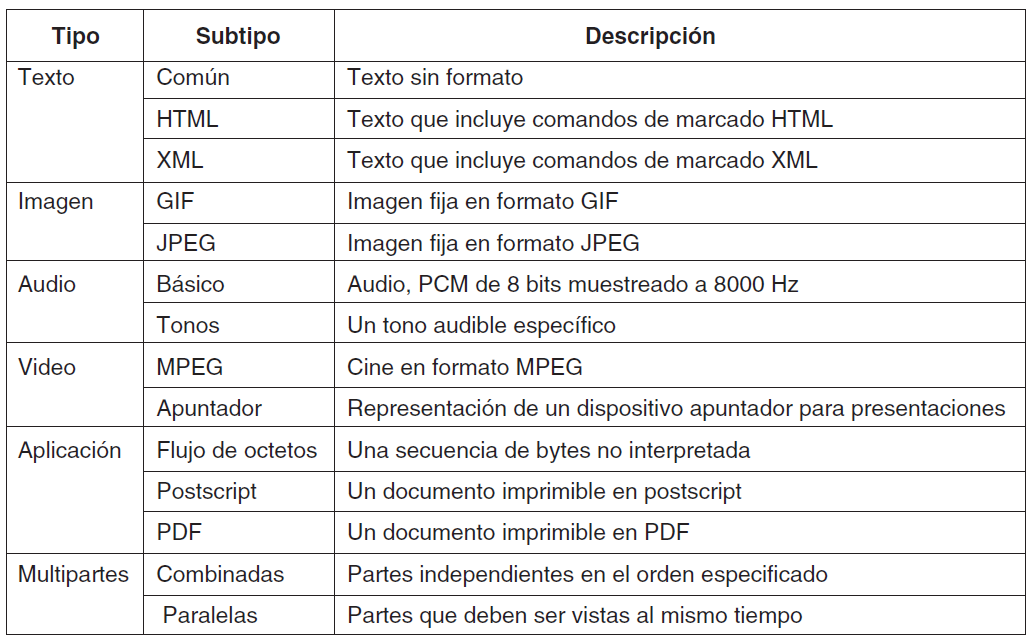
El HTML y el XML también incluyen toda clase de señalizaciones que se refieren a documentos embebidos, es decir, referencias a archivos que deberán ser incluidos para que el documento esté completo. Se puede argumentar que los documentos embebidos transforman un documento web en algo activo. En especial, cuando se considera que un documento embebido puede ser un programa completo ejecutado al vuelo como parte del despliegue de información, no es difícil imaginar lo que se puede hacer.

Los documentos insertados vienen en toda clase de sabores. Esto inmediatamente trae a colación el tema de cómo se pueden equipar los navegadores para manejar los diferentes tipos de formatos de archivo y las maneras de interpretarlos. En esencia, se requieren sólo dos cosas: una forma de especificar el tipo de documento embebido y un modo de permitir que un navegador maneje datos de un tipo específico.

Cada documento (embebido) lleva un tipo MIME asociado. MIME proviene del inglés Multipurpose Internet Mail Exchange —significa intercambio de correo en internet para múltiples usos— y, como su nombre sugiere, originalmente se desarrolló para que informara sobre el contenido del cuerpo de un mensaje enviado como parte del correo electrónico. MIME distingue varios tipos de contenidos de mensaje. Estos tipos también se utilizan en la WWW, aunque la estandarización resulta difícil con los nuevos formatos de datos que aparecen casi a diario.

MIME distingue entre tipos de alto nivel y subtipos. Existe un tipo de aplicación especial para indicar que el documento contiene datos relacionados con una aplicación específica. En la práctica, sólo esa aplicación será capaz de transformar el documento en algo que pueda ser entendido por un ser humano.

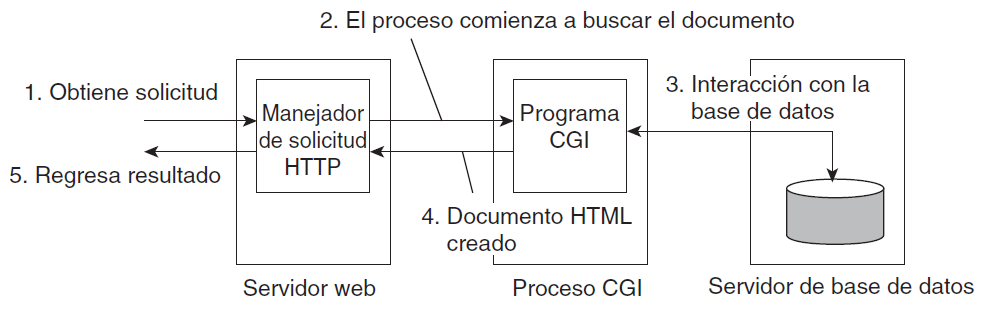
El tipo de múltiples partes se utiliza para documentos compuestos, es decir, documentos constituidos por varias partes donde cada parte desempeña su propio tipo de alto nivel asociado. Por cada tipo de alto nivel, puede haber varios subtipos disponibles. El tipo de un documento se representa entonces en la forma de una combinación de un tipo de alto nivel y un subtipo, tal como, por ejemplo, la aplicación/PDF. En este caso, se espera que sea necesario utilizar una aplicación distinta para procesar el documento, la cual está representada en PDF. Muchos subtipos son experimentales, es decir, utilizan un formato especial que requiere su propia aplicación del lado del usuario. En la práctica, es el servidor web el que la proporcionará, o como un programa aparte que funcionará al lado del navegador, o como un plug-in (componente de software adicional) que pueda ser instalado como parte del navegador.



Arquitecturas de varios niveles

La combinación de HTML (o cualquier otro tipo de lenguaje de marcado tal como XML) con la posibilidad de programación por medio de guiones (scripting) resulta en un medio poderoso para expresar documentos. Sin embargo, apenas si analizamos dónde son procesados en realidad los documentos, y qué clase de procesamiento tiene lugar. La WWW se inició como el sistema cliente-servidor de dos niveles relativamente simple. Por el momento, diremos que esta arquitectura simple se amplió con numerosos componentes para soportar el tipo avanzado de documentos que se acaban de describir.

Una de las primeras mejoras de la arquitectura básica fue soportar la interacción de un usuario por medio de la interfaz de compuerta común, o simplemente CGI (por sus siglas en inglés). La CGI define una forma estándar mediante la cual un servidor web es capaz de ejecutar un programa tomando los datos de un usuario como entrada. En general, los datos de usuario provienen de una forma HTML; ésta especifica el programa a ser ejecutado del lado del servidor, junto con valores de parámetro establecidos por el usuario. Una vez completada la forma, el nombre del programa y los valores de parámetro reunidos son enviados al servidor.



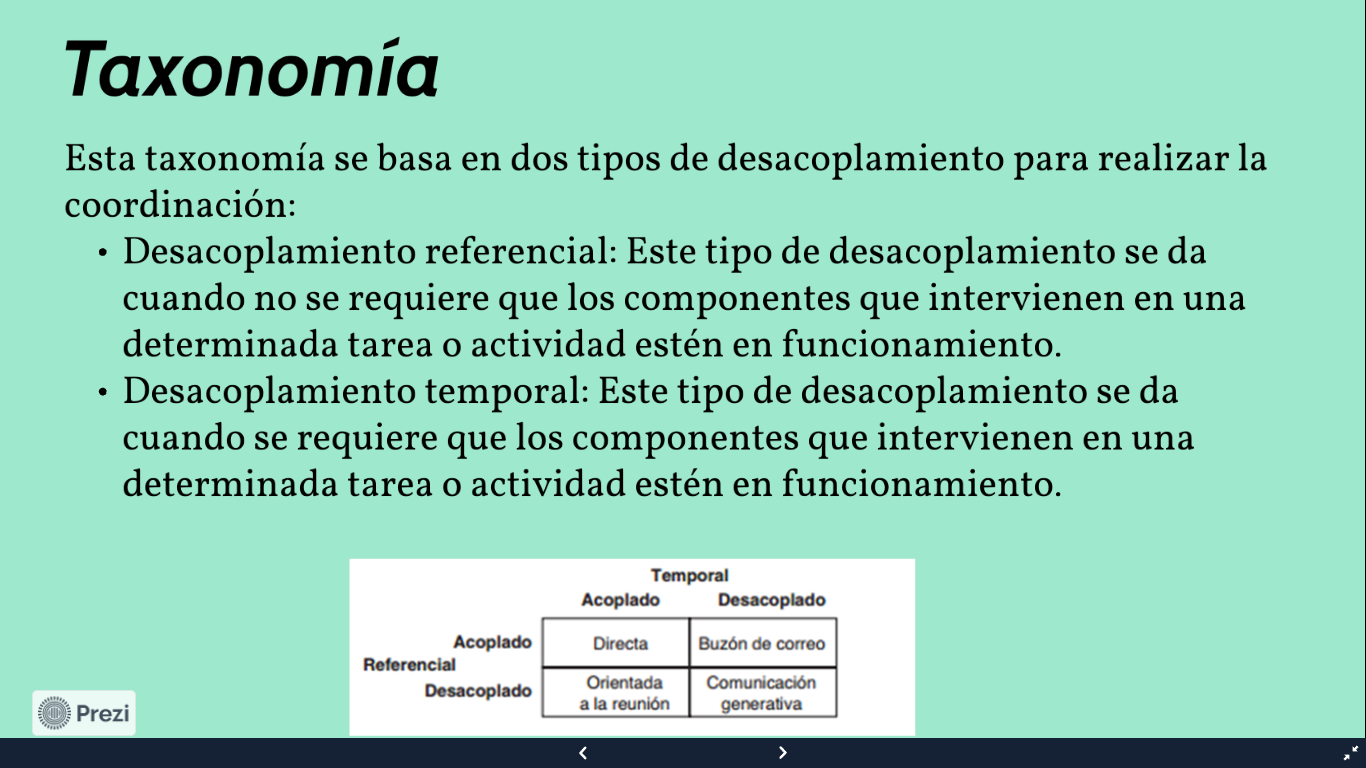
## **6.4 Sistemas distribuidos basados en coordinación**

Este tipo de sistema se basa en la cooperación y la comunicación de los componentes (Servicios, objetos y módulos) del sistema distribuido. Dado a esto los sistemas cuentan con una estructura generalizable la cual se puede aplicar a otros sistemas

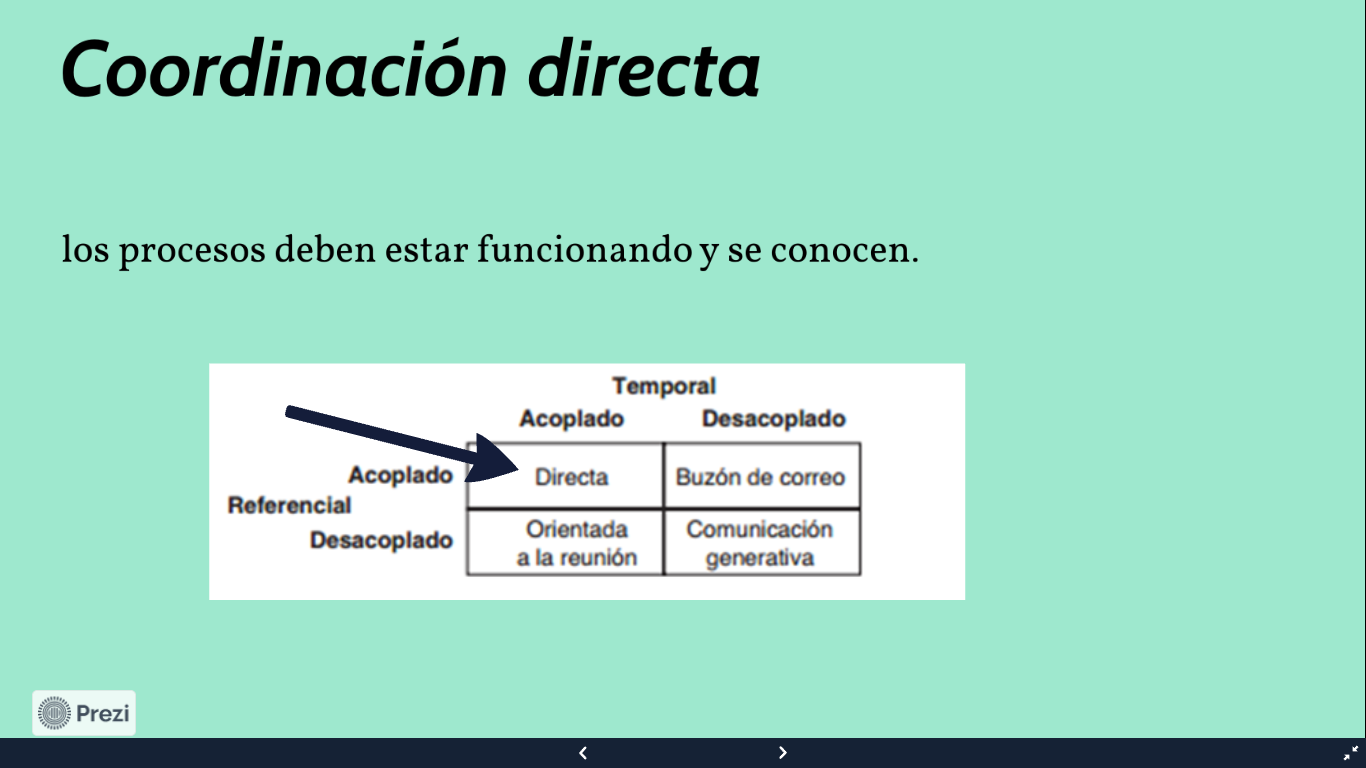
Para la taxonomía encontramos dos formas de representarla, la referencial y la temporal:

Referencial: La referencial la encontramos cuando no se requiere que algún otro componente intervenga en el funcionamiento

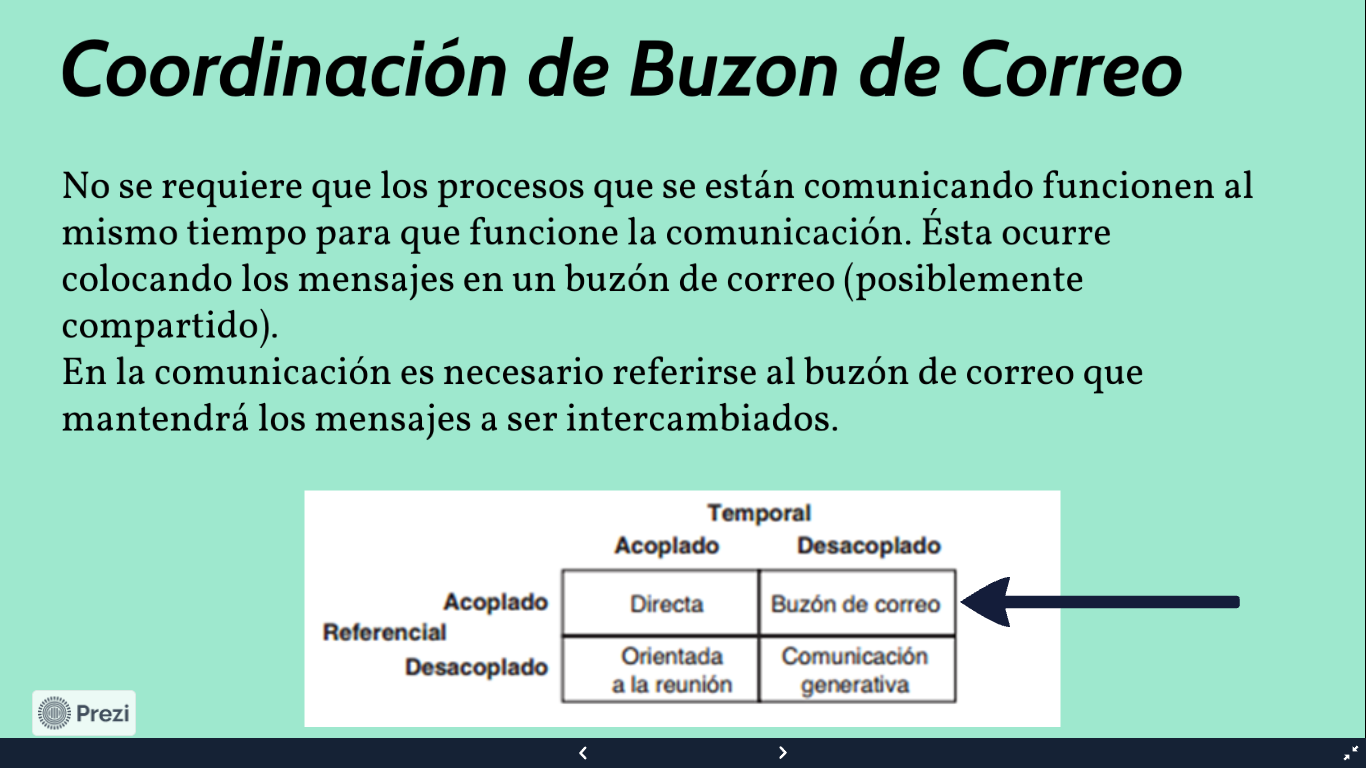
Temporal: Esta, de forma contraria la encontramos cuando los componentes que interfieren con una tarea o actividad, están en funcionamiento



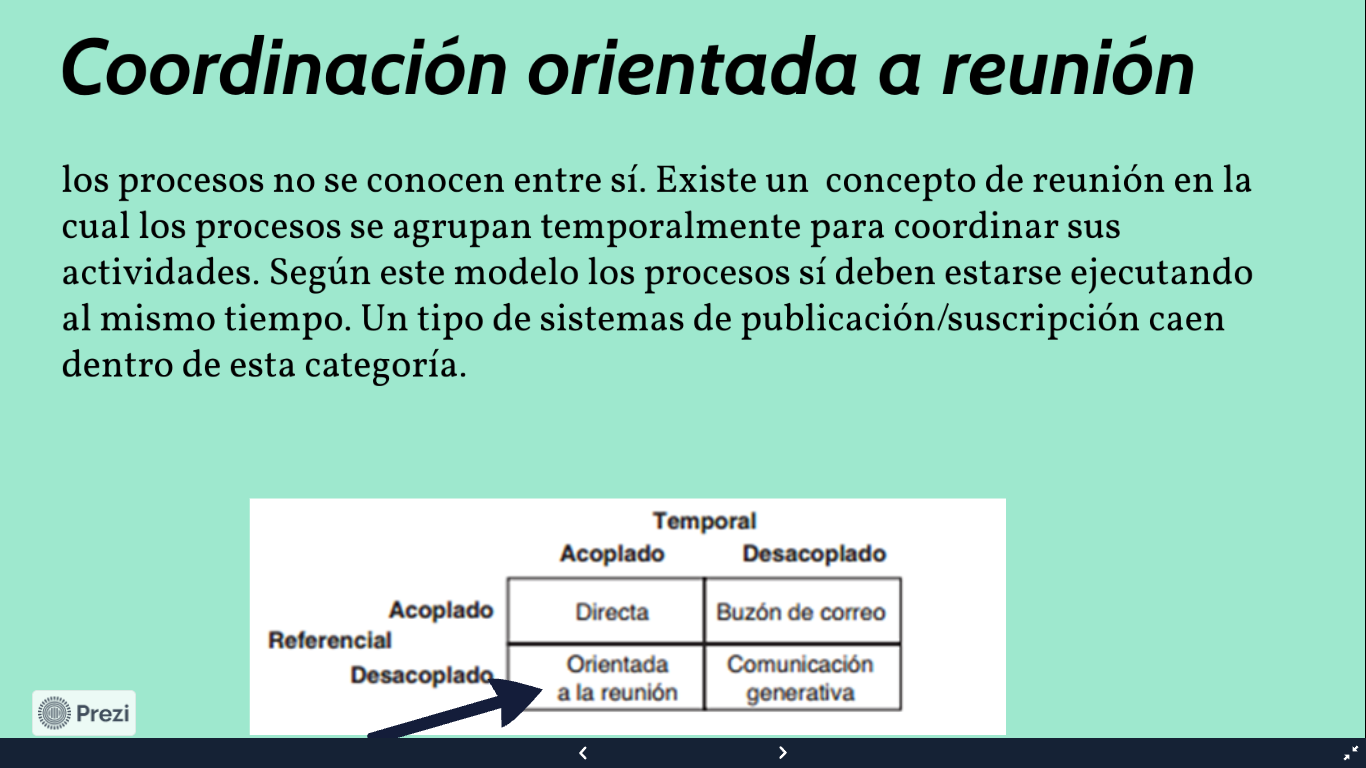
Para la coordinación directa tenemos que tener nuestros funcionamientos trabajando y estos tienen que conocerse



Para la coordinación de buzón de correo estos no necesariamente tienen que los procesos funcionan al mismo tiempo ya que esto ocurre por un uso compartido del correo, el cual permite mantener el mensaje



Dentro de la coordinación orientada a una reunión témenos que estos procesos no son necesarios conocerlo puesto que deberán de estarse ejecutando al mismo tiempo que el resto, para ello servicios de publicación o suscripción están en la misma categoría



Y por último para la comunicación generativa tenemos a todos aquellos procesos que existen en un espacio de datos compartido, pero en el cual no es necesaria la ejecución al unisonó, ya que este se caracteriza por el uso de tuplas ya que estas nos permiten a través de varios campos de entrada identificarlas

Cuando hablamos de la arquitectura de estos sistemas nos enfocamos a aquellas que describen los datos de los elementos a enviar y recibir, estos datos no son identificados en su totalidad por quien lo manda y quien lo recibe, sin embargo, se describen a través de una serie de atributos

Para los elementos se les realiza una anidación de atributos los cuales nos permiten identificar el valor con muchísima más facilidad, estos atributos pueden venir en pares o en demás elementos ya que estos son propios de lo que la información quiera transmitir, pero comúnmente se utiliza el siguiente comparative (atributo, rango) o (atributo, valor). Estas descripciones pueden ser dadas al utilizar varias clases formulados en el mismo atributo

Para los elementos es esencial conocer este tipo de valores ya que Podemos asociarlo con un valor dentro de un vector, como por ejemplo <(a1,v1),(a2v2),(a3v3)> estos claramente tienen que estar dadas por los valores puestos dentro del Sistema (atributo, valor). Estos son importantes ya que proporcionan un gran una forma de construir SD de forma escalable y con un fuerte desacoplamiento

Para ello contamos con sistemas como lo es jini y javaspaces, jini es un Sistema distribuido que implementa un modelo de coordinación de comunicación generative, por lo general se le puede ver relacionado con java, sin embargo, también es posible encontrarlo implementado con otros lenguajes de programación, en cuanto a javaspaces este no es más que un Sistema compartido que guarda las tuplas que representan un conjunto de referencias a objetos de java

Del mismo modo contamos con otro tipo de sistemas para multitransmisión, el sistema TIB/Rendezvous es un sistema que es muchísimo más fácil de identificar, porque en ves de hacer uso de vectores como se mencionaba anteriormente, este hace uso de etiquetas nombradas, es decir que toda la información se encapsula y se etiqueta con una palabra clave que nos sirve para identificar el contenido de este, con ello cada servidor ejecutara un dominio que es el responsable de enviar y entregar los datos de acuerdo al tema

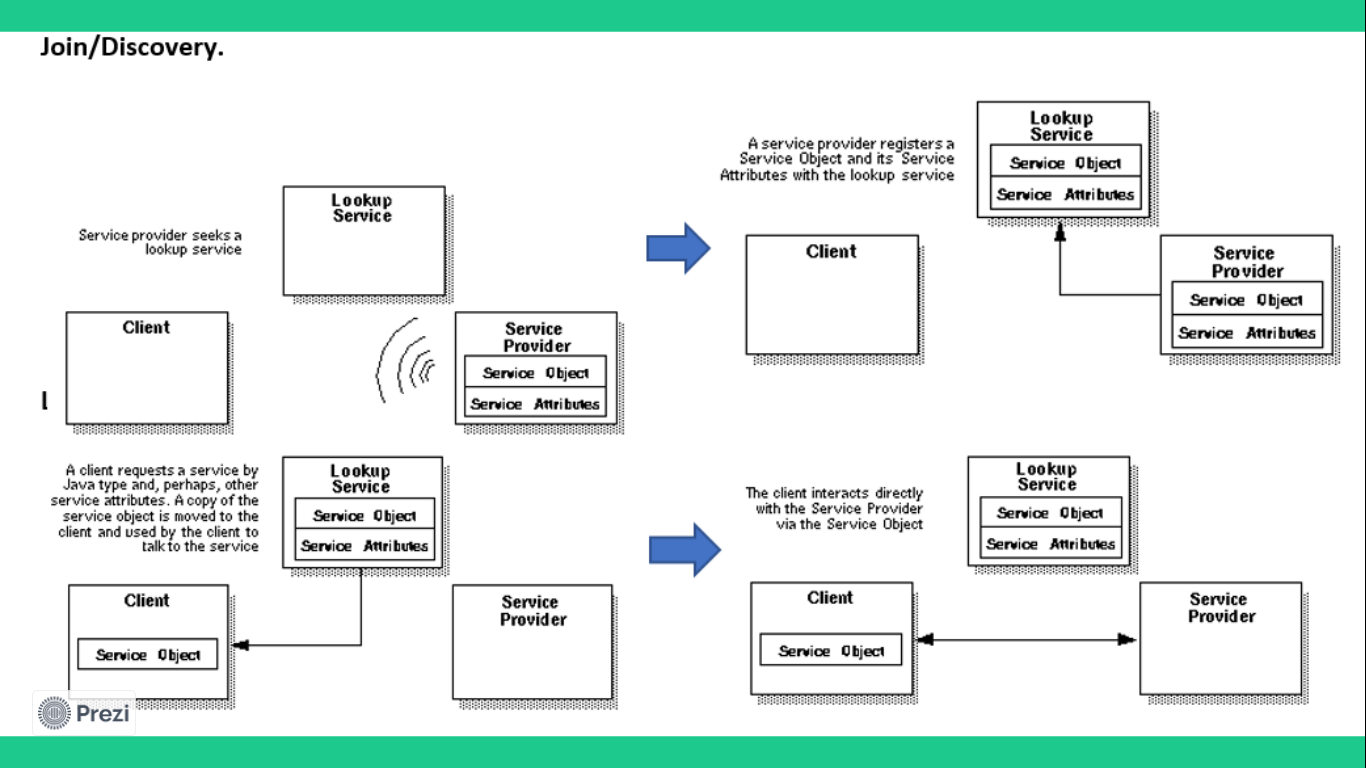
Para los procesos dentro de este sistema de distribución basado en coordinación, contamos con 2 muy importantes.

Por un lado, tenemos el proceso Discovery este es el encargado de realizar un registro de aquellos procesos que se realizan por primera vez, o cuando un servicio es solicitado, lo curioso aquí es que este protocolo cuenta con diferentes versiones como lo es el Protocolo Multicast de Solicitud, el cual se encarga de entrar a una red para localizar un servicio Lookup el cual mostrara los registros.

Por otro lado, tenemos el Protocolo Multicast de Anuncio, el cual permite que el servicio Lookup de un aviso cuando este realiza un alta en la red.

Y por último tenemos el Porotocolo Unicast de Descubrimiento, este ultimo lo que realiza es que si ya se conoce con anterioridad el servicio al cual se trata de registrar, permite tener un acceso más rápido, esto por conocerlo con anterioridad

Ahora por su parte el proceso JOIN es una continuación lógica del protocolo de Discovery y ser ejecuta en el momento de registra un cliente en un servicio de lookup y este se asegura de que el funcionamiento del jini aun si llegan a ocurrir errores dentro del sistema. Para poder realizar esto el sistema le asigna un ID que se utilizara en futuras conexiones y ofrece a su vez un conjunto de atributos actualizables



# **Conclusiones**

Si bien es un tema bastante complejo y completo para estudiar, su dificultad radica mas que nada en entender como es que relacionamos un mensaje desde su transmisor hasta su receptor, existiendo para ello multitud de formas y protocolos para establecer estos, con ello damos al sistema una gran flexibilidad y personalización, tomando solamente recursos que consideremos necesarios, o que realmente nos ayuden a la hora de realizar estos procesos es así que con ello nacen diversas formas de almacenar información, todas de formas remotas y que nos permiten establecer claramente lo que queremos, desde realizar comunicaciones de una computadora a otra, hasta establecer la comunicación de forma virtual a través de un servidor en la web, es así y de este modo que nos permite no solo manipular algunas extensiones de archivo solamente a través de los sistemas de distribución, si no que tenemos el paso abierta para poder utilizar casi cualquiera, esto gracias a los protocolos ideados dentro de la arquitectura creada entorno a estos sistemas, los cuales hoy en día cuentan con una especialización y un trabajo enorme, de modo que es casi mas sencillo entenderlo mediante esquemas que solamente explicado por palabras, ya que estas redes son creadas a partir no solo de un objeto como se puede pensar que es la memoria y listo, si no que cuenta con múltiples elementos que hoy en día son esenciales no solamente para su funcionamiento si no también para la seguridad de estos, cosa que se a complicado a medida que los sistemas se desarrollan mas y mas

# **Herramientas y recursos**

* Imágenes de Google
* Power Point para crear la exposición
* Youtube
* El documento de nuestra investigación del tema
* Scribbr para el APA
* Equipo de cómputo.

# **Bibliografía**

*Sistemas de archivos distribuidos - ¿Qué es DFS?* (s. f.). Nutanix. <https://www.nutanix.com/es/info/distributed-file-systems>

Tanenbaum, A. S. & Steen, V. M. (2006). *Distributed Systems: Principles and Paradigms* (2nd ed.). Pearson Prentice Hall.

UNIDAD 4: LA TECNOLOGÍA DE OBJETOS DISTRIBUIDOS. (2012, junio 16). iaclaudia. <https://iaclaudia.wordpress.com/2012/06/16/unidad-4-la-tecnologia-de-objetos-distribuidos/>

Sanchez, J. C. L. (s. f.). *Sistemas Distribuidos basados en coordinacion*. prezi.com. https://prezi.com/p/ooqsbqfe6gkb/sistemas-distribuidos-basados-en-coordinacion/