**Sistemas operativos distribuidos:**

Cuestiones para implementar un Sistema Distribuido:

* ¿Cómo administrar los recursos compartidos?

Administración de recursos

* ¿Cómo acceder a recursos sin conocer su ubicación? -> mediante nombres globales

\* nombres globales:

* Se busca que la identificación de un recurso sea independiente de su ubicación fisica.

En lugar de //Compuradora/NombreRecurso se usa //NombreGlobalRecurso

* Para lograr se usan <<servidores de nombre>>:

- poseer una tabla de asociación (nombre global -> pasa a ubicación y nombre local)

- estar replicando para evitar “cuello de botella” y mejorar la disponibilidad

* ¿Cómo mejorar el acceso de recursos remotos? -> mediante transferencia masiva y cache de datos

\*transferencia masiva datos:

* en lugar de enviar solo los datos solicitados, se envían más datos (la mayoría o todo)
* se busca el ‘overhead’ generado por los protocolos de comunicación
* requiere que los datos no solicitados se almacenen localmente para poder accederlos posteriormente (caches datos)

\*cache de datos:

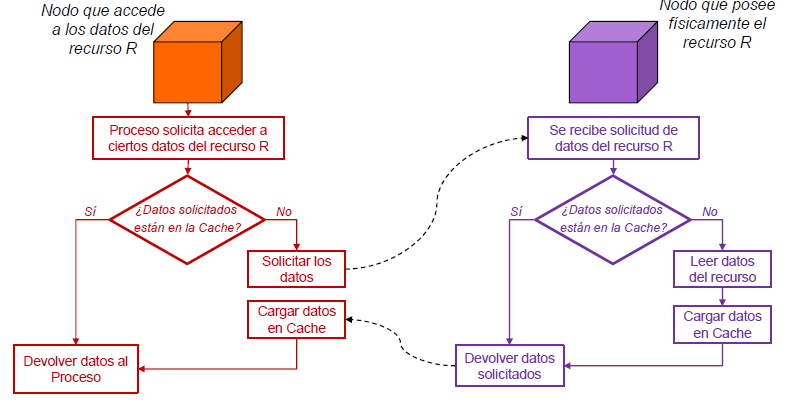
* se utiliza para facilitar el acceso a información que:
* ha sido recibida (solicitada o no)
* se debe enviar
* se puede almacenar temporalmente en memoria y/o disco local
* dificultad: mantener la << cache consistente >>

*- ¿cómo y cuándo acceder a la información de la cache?*

*- ¿cómo y cuándo actualizar la información en la cache y/o recurso?*

*- ¿cómo mantener la cache consistente?*

* Acceso a la cache:



* Actualización de la cache y recurso:

-determina en que momento las actualizaciones se escriben en la cache y/o en el recurso

-dos posibilidades de actualización:

->escritura inmediata (proceso necesita actualizar recurso, el so actualiza cache local, manda la info al sist op que tiene el recurso, ese sist op actualiza el cache local y la información del recurso).

->escritura demorada (proceso solicita a su sist op actualizar un recurso, este actualiza la copia local de su cache y le dice a su proceso que ya está actualizado, pero todavía no le mando el msj al otro sist op para que lo actualice), por ende

->cada cierto tiempo

->finalizar el uso del recurso

Ahí le va a mandar todas las actualizaciones.

* Consistencia de la cache:

-se considera consistente si posee los mismos datos que el recurso

-se utiliza timestamps para comparar los datos reales vs datos cacheados

-para notificar los cambios se puede usar un esquema:

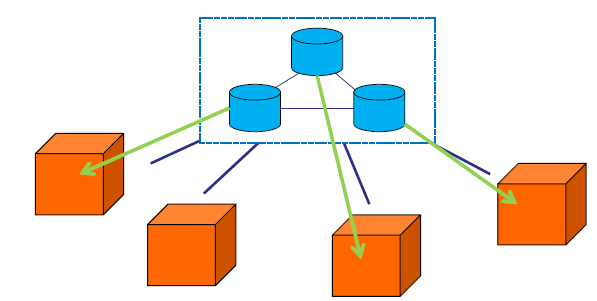
-> centralizado

->distribuido

->jerárquico

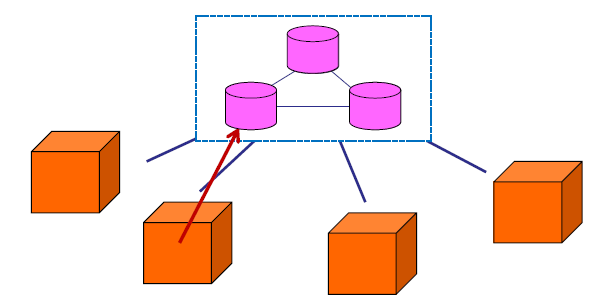
\*replicación de recursos:

* Se ocupa de manejar copias de los recursos que se pueden acceder en forma concurrente y transparente
* Aumenta la disponibilidad y el rendimiento del sistema



* El problema surge al intentar mantener la consistencia entre las copias cuando algún proceso modifica una de ellas

->se debería lograr que todos los recursos estén actualizados



Por ende, para dicho problema se utiliza lo siguiente:

* Algoritmos para el manejo de las copias:

-garantiza que se accede siempre al resultado de la última modificación

-busca mantener la consistencia usando métodos

+transparentes (usuario y procesos)

+escalables (funcionan para un sist distribuido por pocos o muchos nodos)

+eficientes (por si el sist continuamente tiene que estar actualizando las copias, puede ser un problema en el rendimiento de ejecución)

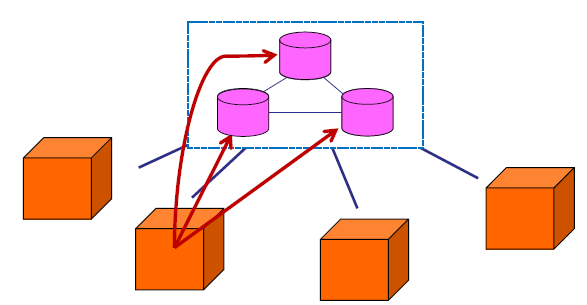
-basados en el uso de:

+versionado de datos (etiquetas de tiempo)

+transacciones

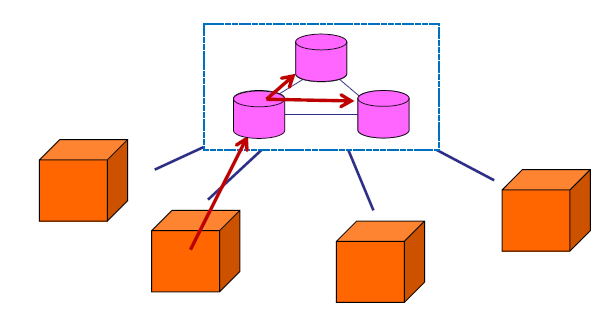
1. Actualiza todo o nada:

Cuando un proceso necesita actualizar un recurso, solicitara la modificación a todas las copias (todas se actualizan al mismo tiempo), el problema que esto implica mucho tiempo. No es el más recomendado.



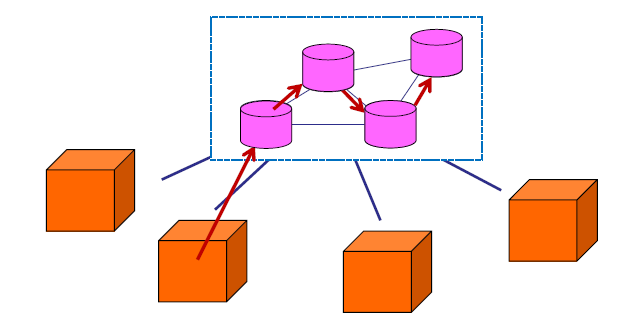
1. Primera copia:

Nuestro nodo que quiere modificar la información de cierto recurso, modificara la copia que está utilizando, y el nodo de este se ocupara de transmitir las modificaciones a las otras copias. Sigue siendo lento su proceso.



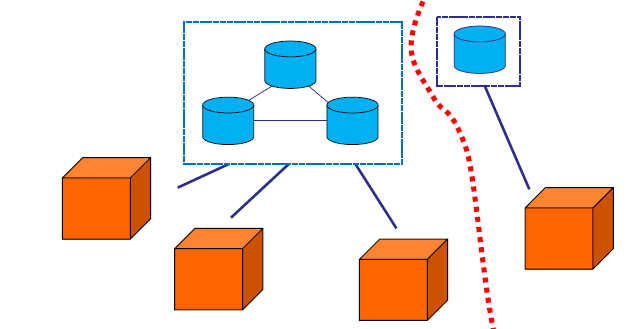
1. Gossiping:

El nodo donde está el proceso, le mandara la actualización a la única copia que conoce, esta se las mandara a otras y de estas a otras más, no habrá un solo nodo que logre que todas las copias este actualizadas, sino se aplique un algoritmo más distribuido, centralizado y más transparente. Se garantiza que, en algún momento, todas las copias quedaran todas actualizadas.



1. ¿Qué sucede cuando la red se particiona?:

Nuestra red tiene un desperfecto, y una copia queda afuera. (un recurso tiene 4 copias y una de esas quedas afuera de la comunicación de las otras copias). No se



Puede actualizar y se necesita leerla y escribirla (la info de esa copia se pueda acceder de forma confiable y actualizada)

1. Manejo del quorum:

-Las operaciones son permitidas solo cuando se obtiene un consenso de la mayoría de los nodos disponibles.

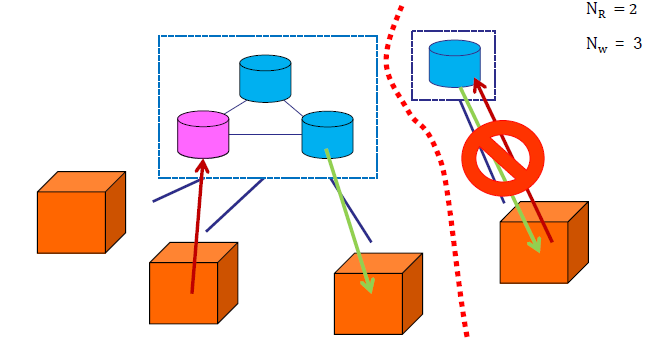
-Se define una cantidad mínima de nodos que deben aprobar la operación a realizar; si se obtiene la aprobación, esta se realiza; caso contrario se rechaza.

La cantidad mínima puede ser diferente para lecturas (NR) y escrituras (nw) donde

Nr + nw > ntotal

Nr <\_ nw

Nw > ntotal / 2

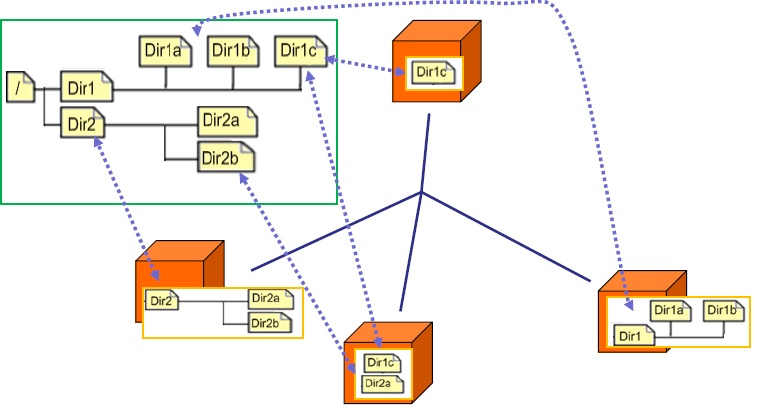


Además de los nombres globales, transferencia masiva y cache datos, replicación de recursos.

Están:

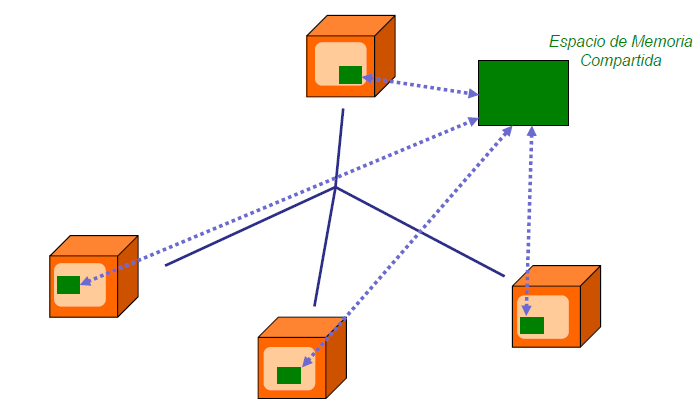
* Sistema de archivos distribuidos

Permite generar un sistema de archivos global compartido entre los nodos de un sistema distribuido



* Memoria distribuida

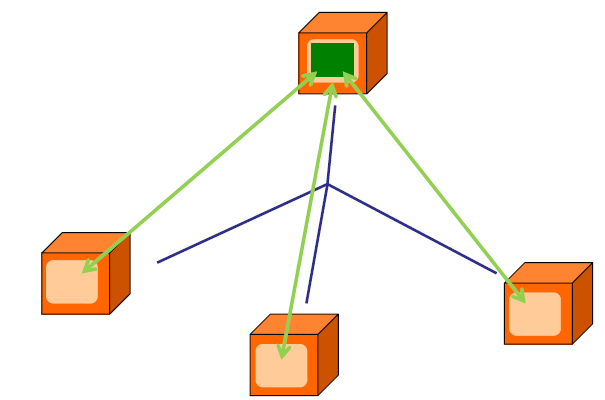
Permite generar secciones de memoria compartida entre los nodos de un sistema distribuido



Dentro de la memoria distribuida, tiene posibilidades de implementación:

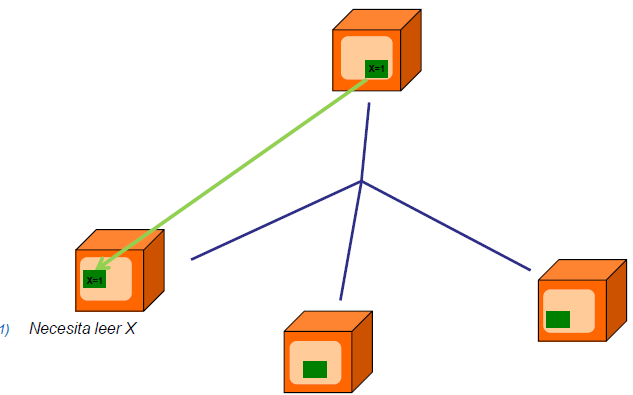
* Centralizado->

Toda la info compartida estará almacenada en un único nodo, y cada vez que un proceso de otro nodo quiere acceder a dicha info, lo hará mediante mensajes.

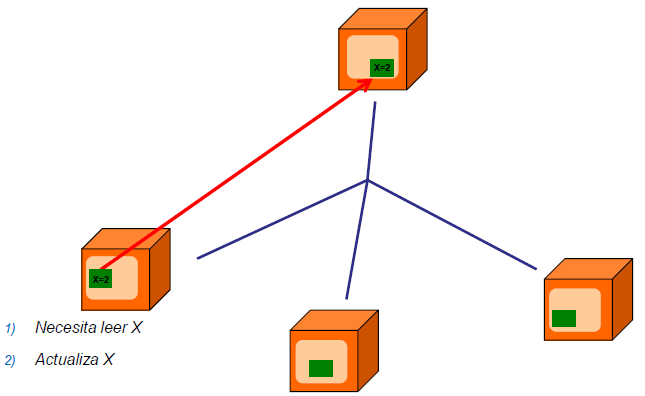


* Descentralizado con replicación:

La información real está distribuida en todos los nodos, cuando se necesita acceder a cierto valor de cierta variable, el nodo (n2) que necesita la info (n1), se replica el dato de la variable (x=1)

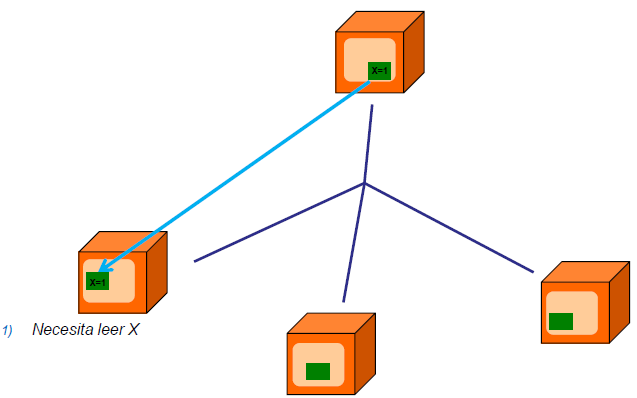


Si se necesita actualizar, primero se copia el nuevo valor de forma local (n2->x=2), y eso se replica al nodo original (n1->x=2), de manera que siempre la info se acceda de manera local. Como problema debo gestionar las distintas copias y a las que no necesito debo borrarlas para que no se desperdicie la memoria RAM.

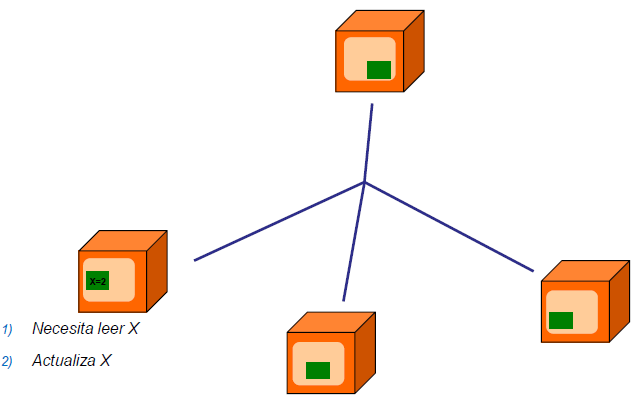


* Descentralizado con migración:

Parecido al anterior, yo tengo el nodo1 en la variable 1 (n1->x=1) y el nodo2 necesita acceder a esa variable, se migrará esa variable con su valor del nodo1 al nodo2 (hacemos una copia, y ponemos n2->x=1, pero en el nodo1 se borrará dicha variable).



Cuando se tiene que actualizar, solo se va actualizar el nodo que tiene la variable (n2->x=2) y (n1 ya no tiene la variable x).



Descentralizado con migración, también facilita la gestión de la mutua exclusión, porque solamente podrá modificar los datos el nodo que tiene la copia local. (trabaja muy pareció al mecanismo fortokens, este sería la variable que uno quiere gestionar). Tendríamos 1 token por cada variable y cada sincronización se podría manejar de forma independiente.