SOLUCIONES ACTIVIDADES UNIDAD 7

1.- La **ocultación** implica ocultar las diferencias de los ordenadores del sistema (ej: diferentes sistemas operativos, diferentes arquitecturas de máquina, diferentes tipos de recursos...). También implica que el usuario no perciba la complejidad de los mecanismos de comunicación que se necesiten para la cooperación de las actividades de la aplicación.

En los tres ejemplos, cuando nos conectamos al sistema no nos importa (ni podemos ver) sobre qué máquinas, sistemas operativos, etc. se está ejecutando la aplicación.

El **acceso homogéneo** implica que los usuarios y aplicaciones interactúen con el sistema de una manera uniforme, con independencia del ordenador concreto que se emplee.

En los tres ejemplos, con independencia de dónde nos conectemos, obtendremos un acceso similar a la aplicación.

La **escalabilidad** implica no que no resulte difícil incorporar más ordenadores para atender a un mayor número de usuarios.

En los tres ejemplos, se podrán incorporar servidores para poder atender al incremento de demanda de usuarios.

Finalmente, la **disponibilidad** implica que los servicios ofrecidos por un sistema estén siempre disponibles.

Tanto PoliformAt como los ordenadores para la docencia en los laboratorios (ej: el entorno Virtual) suelen funcionar generalmente sin problemas, aunque se produzcan fallos en sus servidores. Como mucho, en algunos momentos se puede observar cierto retraso en la respuesta.

En los servicios de mensajería en internet, aunque fallen algunos servidores intermedios, sabemos que el mensaje que enviamos siempre llegará a su destino.

2.- a) transparencia de ubicación

b) transparencia de replicación

c) transparencia de migración, transparencia de reubicación

d) transparencia de acceso

3.- a) Escalabilidad administrativa

b) Escalabilidad de tamaño

c) Escalabilidad de distancia

d) Escalabilidad de tamaño

e) Escalabilidad de tamaño, escalabilidad de distancia

4.- *A) Laboratorio de prácticas // Sistema de ficheros distribuido.*

El Laboratorio completo y el sistema de ficheros distribuidos proporcionan básicamente las mismas características (la posibilidades que ofrece el laboratorio descansan sobre el sistema de ficheros distribuido), así que se discuten de forma conjunta. Los tipos de transparencia que requieren son:

Acceso: sí. Utilizamos los mismos mecanismos de acceso con independencia de si se trata de un fichero local o remoto, o de cómo se representa internamente.

Ubicación: sí. Accedemos a los ficheros como si fuesen recursos locales, y no nos preocupa su ubicación real

Migración: no se sabe. El enunciado no aporta información suficiente para contestar de forma categórica

Reubicación: no se sabe. El enunciado no aporta información suficiente para contestar de forma categórica

Replicación: sí. Puede que un fichero o partes del mismo esté replicado (ej. en el servidor y en uno o más clientes), y el usuario no necesita conocer esos detalles.

Concurrencia: sí. Es posible que varios usuarios accedan de forma simultánea al mismo recurso, y el sistema debe garantizar que no se producen interferencias.

Fallos: debería soportarla, aunque el enunciado no aclara lo suficiente (ej. no sabemos si el servidor remoto está replicado o no)

Persistencia: sí. El uso de cachés es transparente para el usuario

B) Sistemas de computación en Grid

Acceso: sí. Oculta diferencias en la representación de los datos y el modo de acceso a los mismos.

Ubicación: sí. Oculta dónde se ubican los recursos.

Migración: sí. La descripción no lo aclara, pero normalmente sí (facilita el objetivo de intercambio de datos o recursos de computación)

Reubicación: no se sabe. El enunciado no aporta información suficiente para responder

Replicación: normalmente sí (para mejorar la escalabilidad y la tolerancia a fallos), aunque el enunciado no lo aclara de forma explícita

Concurrencia: sí. Necesaria, ya que es previsible el acceso concurrente a las mismas entidades.

Fallos: sí. Necesaria, ya que cuanto mayor es el tamaño del sistema mayor es la probabilidad de que falle alguno de sus componentes, y eso no debe afectar a los clientes .

Persistencia: sí. Necesaria. Por razones de eficiencia, el uso de cachés es muy frecuente, y debe resultar transparente al usuario.

C) Sistemas de información transaccionales

Acceso: sí. Oculta diferencias en la representación de los datos y el modo de acceso a los mismos.

Ubicación: sí. Oculta dónde se ubican los recursos.

Migración: no se sabe. La descripción no es suficientemente detallada para responder.

Reubicación: no se sabe. La descripción no es suficientemente detallada para responder.

Replicación: sí. En la descripción se indica el uso de técnicas de replicación.

Concurrencia: sí. Garantizan acceso simultáneo libre de interferencias.

Fallos: sí. En la descripción indica que debe seguir funcionando en caso de fallo

Persistencia: sí. Accedemos a información definida como estable sin necesidad de conocer el medio exacto en el que se almacena en cada instante (ej. Memoria volátil o persistente)

5.- Características de un algoritmo descentralizado:

A) Ningún nodo tiene toda la información completa que requiere el algoritmo

B) Los nodos únicamente toman decisiones en base a su conocimiento local

C) El fallo de un nodo no impide que el algoritmo progrese

D) No se asume la existencia de un reloj físico global

El primer algoritmo no es descentralizado, ya que no cumple todas las condiciones. Ej. No cumple con "A" (pues el coordinador conoce el estado de la sección crítica y controla el acceso a la misma) ni tampoco con "C" (el fallo del coordinador impide el progreso del algoritmo). El principal inconveniente del primer algoritmo es la falta de escalabilidad en tamaño (el coordinador es un cuello de botella, ya que todos los clientes contactan con él), y el hecho de que el coordinador se convierte en un punto de fallo único (si cae el nodo coordinador, el algoritmo no puede progresar). Este último problema se podría solucionar si este algoritmo se complementa con un algoritmo de elección de líder, que permita que, en caso de caer el nodo coordinador, se pueda elegir a un nuevo nodo que actúe de coordinador. Esta solución implicaría que todos los nodos deben tener el código necesario para actuar, en su momento, tanto de coordinadores como de clientes.

Respecto al segundo algoritmo, es mucho más descentralizado, ya que aunque todos los nodos conocen los identificadores de los otros hilos, ninguno de ellos tiene la información completa sobre qué nodo (qué hilo) está en la sección crítica, sobre quiénes desean entrar en esa sección crítica (y, por tanto, están esperando). En el algoritmo anterior, toda esta información recaía en el coordinador. Además, los nodos toman decisiones en base a su conocimiento local. Esta propiedad implica que con la información parcial que consigan del problema les basta para tomar sus decisiones. Para ello pueden intercambiarse mensaje con otros. Lo único que se requiere en esta segunda propiedad es que cada nodo pueda progresar con la información que tenga disponible y que tal progreso no se vea amenazado por la indisponibilidad momentánea de alguno de los participantes. En este caso, cuando desean entrar en la sección crítica simplemente envían un mensaje TRY y esperan a que los demás les contesten. En el enunciado se indica que todos los hilos saben qué otros hilos existen en el sistema en cada momento y que actualizan sin problemas esa información en caso de fallos. Por tanto, el algoritmo podrá además progresar en caso de que un

nodo falle, pues el resto de nodos detectará dicho fallo y, en su caso, ya no tendrán que esperar a la contestación de dicho nodo.

Finalmente, los nodos no requieren de emplear sincronización de los relojes, pues para abordar un empate en caso de varios hilos que necesiten entrar a la vez en la sección crítica, se utiliza el número de identificador para desempate (ganando el identificador más alto). Por tanto, este segundo algoritmo es más descentralizado que el primero, aunque requiere intercambiar muchos más mensajes y sólo tendría sentido cuando la difusión de mensajes es eficiente (ej. sistemas en bus).

Para facilitar su descentralización, el segundo algoritmo se debe complementar con algoritmos para la detección de la caída de algún nodo y la actualización de esta información al resto de nodos. En caso de caída de algún nodo, los nodos que deseaban entrar en la sección crítica (y que ya habían enviado su mensaje TRY al resto) deberán repetir su solicitud, pues podría haber caído justamente el nodo que controlaba la sección crítica y, en ese caso, el mensaje de OK que esperaban de dicho nodo no se recibiría nunca.

6.- Un middleware es una capa de software, por encima del sistema operativo pero por debajo del programa de aplicación, que proporciona una abstracción común de programación en un sistema distribuido, ofreciendo transparencia, escalabilidad y facilitando el acceso remoto y, si se utilizan estándares, la apertura del sistema.

Ejemplos de middleware: Java RMI, Java Message Service, DNS