**Acordeón**

Definición de sistema distribuido de acuerdo al libro:

Un sistema distribuido es aquel en el cual los componentes de hardware o software localizados en una red de computadoras, coordinan sus acciones solo a través de mensajes. Algunos ejemplos de aplicaciones distribuidas son:

* Búsqueda web
* Juegos multiplayer
* Sistemas de mercado financiero
* Email, redes sociales, eCommerce

**Consecuencias de un sistema distribuido** (no sé porque dice consecuencias, más bien parecen retos a superar)

* Concurrencia: Esto se refiere a la ejecución concurrente de programas y la capacidad de compartir recursos entre varias computadoras sin que el trabajo de ninguno de los elementos se vea afectado
* No global clock (no supe de qué forma traducirlo): debido a la naturaleza de un sistema distribuido la coordinación precisa es esencial y depende de que todos los elementos en la red de trabajo tengan una idea compartida del tiempo en el que las acciones del programa ocurren, sin embargo, existen muchos inconvenientes sobre la precisión con la que las computadoras de una red de trabajo pueden sincronizar sus relojes.
* Fallas independientes: Todos los sistemas pueden fallar y siempre existirán nuevas formas en las que pueden hacerlo, sin embargo, que una computadora falle no significa que todo el sistema falle y cuando alguna parte del sistema falla no es inmediatamente dado a conocer a todos los componentes con los que se comunica, es decir, cada componente puede fallar, dejando a los otros continuar funcionando/ejecutando su tarea.

**Desafíos para los sistemas distribuidos**

* Heterogeneidad: Uno de los desafíos mas grandes es lograr protocolos de comunicación comunes ya que existen muchas interfaces y arquitecturas diferentes, el reto es lograr diseñar estándares comunes para que cualquier componente se pueda integrar a una red de trabajo.
* Middleware: el middleware nació en un principio como solución para conectar software nuevo con versiones antiguas, sin embargo, con la aparición de las tecnologías en red, se convirtió en una solución para los programadores de servidores y aplicaciones distribuidas soslayando los problemas de la heterogeneidad proporcionando un modelo computacional uniforme su alcance.

Ejemplo CORBA: es un estándar que permite que diversos componentes de software escritos en múltiples lenguajes de programación y que corren en diferentes computadoras, puedan trabajar juntos; es decir, facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos.

* Heterogeneidad y código móvil: el termino código móvil puede emplearse para referirse al código que puede ser enviado desde un computador a otro y ejecutarse en este, por eso los applets de java son un ejemplo de ello. La aproximación de maquina virtual provee un modo de crear código ejecutable sobre cualquier hardware: el compilador de un lenguaje concreto genera código para una maquina virtual en lugar de código apropiado para hardware particular, mientras que el compilador de java produce código para la maquina virtual java, la cual solo necesita ser implementada una vez para cada tipo de maquina con el fin de lanzar programas java.

**Extensibilidad**

La extensibilidad de un sistema de cómputo es la característica que determina si el sistema puede ser extendido y reimplementado en diversos aspectos. La extensibilidad de los sistemas distribuidos se determina en primer lugar por el grado en el cual se pueden añadir nuevos servicios de compartición de recursos y ponerlos a disposición para el uso por una variedad de programas cliente.

* Los sistemas abiertos se caracterizan porque sus interfaces están publicadas.
* Los sistemas distribuidos abiertos se basan en la providencia de un mecanismo de comunicación uniforme e interfaces publicas para acceder a recursos compartidos
* Los sistemas distribuidos abiertos pueden construirse con hardware y software heterogéneo, posiblemente de diferentes proveedores. Sin embargo, la conformidad con el estándar publicado de cada componente debe contrastarse y verificarse cuidadosamente si se desea que el sistema trabaje correctamente.

**Seguridad**

La seguridad tiene tres componentes: confidencialidad (protección contra el descubrimiento por individuos no autorizados), integridad (protección contra la alteración o corrupción), disponibilidad (protección contra interferencia con los procedimientos de acceso a los recursos)

La seguridad no solo consta en mantener la confidencialidad de un mensaje con información sensible a través de internet, sino que también se refiere a garantizar la identidad de las personas que proveen y reciben información.

**Escalabilidad**

Se dice que un sistema es escalable si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios. Retos en cuanto a escalabilidad:

Control del coste de los recursos físicos: Según crece la demanda de un recurso, debería ser posible extender el sistema, a un coste razonable, para satisfacerla.

Control de las pérdidas de prestaciones: En un sistema siempre hay perdida de recursos de procesamiento cada vez que se aumenta la demanda, sin embargo, deben encontrarse soluciones para conseguir que esta perdida de recursos sea mínima, ante cualquier situación. (esta parte no supe bien como resumirla, si quieren échenle un ojo en la página 17 segunda parte xd).

Evitar cuellos de botella: Para evitar cuellos de botella los algoritmos deberían ser descentralizados.

Idealmente, el software de sistema y aplicación no tiene por qué cambiar cuando la escala del sistema se incremente, pero esto es difícil de conseguir.

**Tratamiento de fallos**

Los fallos en un sistema distribuido son parciales; es decir, algunos componentes fallan mientras otros siguen funcionando. Consecuentemente, el tratamiento de fallos es particularmente difícil.

Detección de fallos

* Enmascaramiento de fallos: Algunos fallos detectados pueden ocultarse o atenuarse. Ejemplos:
  + Los mensajes pueden retransmitirse cuando falla la recepción
  + Eliminar un mensaje corrupto
* Tolerancia de fallos: La mayoría de sistemas tienen fallos que posiblemente no son prácticos de detectar y ocultar, los sistemas pueden diseñarse para tener cierta tolerancia a los fallos, por ejemplo, cuando un cliente no puede conectarse a un servidor, el servidor le informa del problema y le sugiere intentarlo más tarde.
* Recuperación frente a fallos: La recuperación implica el diseño de software en el que, tras una caída del servidor, el estado de los datos pueda reponerse o retractarse a una situación anterior.
* Redundancia: Puede lograrse que los servicios toleren fallos mediante el empleo redundante de componentes. Ejemplos:
  + Siempre deberá haber al menos 2 rutas diferentes cualesquiera
  + Una base de datos puede encontrarse replicada en varios servidores para asegurar que los datos siguen siendo accesibles tras el fallo de cualquier servidor.

**Concurrencia**

Tanto los servicios como las aplicaciones proporcionan recursos que pueden compartirse entre los clientes en un sistema distribuido. Existe por lo tanto la posibilidad de que varios clientes intentes acceder a un recurso compartido a la vez.

Cada objeto que represente un recurso compartido en un sistema distribuido debe responsabilizarse de garantizar que opera correctamente en un entorno concurrente.

Para que un objeto sea seguro en un entorno concurrente, sus operaciones deben sincronizarse de forma que sus datos permanezcan consistentes. Esto puede lograrse mediante el empleo de técnicas conocidas como semáforos.

**Transparencia**

Se define transparencia como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes en un sistema distribuido, de forma que se perciba el sistema como un todo mas que como una colección de componentes independientes. Formas de transparencia:

* Transparencia de acceso, que permite acceder a los recursos locales y remotos empleando operaciones idénticas
* Transparencia de ubicación que permite acceder a los recursos sin conocer su localización
* Transparencia de concurrencia que permite que varios procesos operen concurrentemente sobre recursos compartidos sin interferencia mutua
* Transparencia de replicación que permite utilizar múltiples ejemplares de cada recurso para aumentar la fiabilidad y las prestaciones sin que los usuarios y los programadores de aplicaciones necesiten su conocimiento
* Transparencia frente a fallos que permite ocultar los fallos, dejando que los usuarios y programas de aplicación completen sus tareas a pesar de los fallos del hardware o de los componentes de software.
* Transparencia de movilidad que permite la reubicación de recursos y clientes en un sistema sin afectar la operación de los usuarios y programas.
* Transparencia de prestaciones que permite reconfigurar el sistema para mejorar las prestaciones según varia su carga.
* Transparencia al escalado que permite al sistema y a las aplicaciones expandirse en tamaño sin cambiar la estructura del sistema o los algoritmos de aplicación.