Ingeniería de Servidores

Apuntes de Teoría

INGENIERÍA INFORMÁTICA

2023

ÍNDICE GENERAL

CAPÍTULO

0 NOTACIÓN

Estos apuntes están realizados con las diapositivas del profe Hector Emilio Pomares Cintas y apuntes de clases. Si no has podido asistir o estás estudiándotelos la semana antes del examen: mucha suerte y espero que te ayuden.

Estas son las diferentes notaciones que encontrarás en los apuntes:

0.1 Nota

Nota **Notación**

Una nota utilizada para aclarar cosas que se han dicho en clase sobre temas específicos.

0.2 Fórmula

Fórmula 0.2.1 **Una fórmula**

Esto es una fórmula para ejercicios del examen.

0.3 Definición

Definición 0.3.1 Una palabra cualquiera
Aquí vendría el texto definiendo.

0.4 Listas

Listas enumeradas

1. Primer item
2. Segundo item
3. Tercer item

Listas con puntos

- Primer item
- Segundo item
- Tercer item

CAPÍTULO 1

INTRODUCCIÓN A LA INGENIERÍA DE SERVIDORES

1.1 ¿Qué es un servidor?

Definición 1.1.1 Sistema Informático (S.I.)

Es la combinación de **hardware** (componentes físicos, como cables), **software** (componentes lógicos, como el sistema operativo) y **peopleware** (recursos humanos, como el personal técnico que instala el sistema).

Hay muchas formas de clasificar un sistema informático, por ejemplo podríamos hacerlo según su paralelismo de datos e instrucciones, como vimos en AC (SIMD, MISD...). Aquí hay clasificaciones más generales:

S.I según su uso

- **Uso general**: Son utilizados por un usuario para ejecutar aplicaciones, como un portátil.
- **Uso específico**: Son utilizados para propósitos específicos, como los **sistemas empotrados**, de una lavadora o los **servidores**, que sirven para proporcionar servicios.

Definición 1.1.2 Sistema empotrado

Sistemas informáticos **acoplados** a otro aparato, diseñados para realizar una o algunas funciones específicas, con restricciones de tamaño, tiempo de respuesta (tiempo real para un banco o cámaras de seguridad), consumo y coste. Suelen utilizar **microcontroladores**.

Definición 1.1.3 Microcontrolador

Procesador, memoria e interfaces de comunicación en un ordenador muy pequeño diseñado para sistemas restrictivos.

Definición 1.1.4 Servidores

Son sistemas informáticos que **proporcionan servicios** a otros SI llamados **clientes**. Deben **pertenecer a una red** mediante la cual se conecta a sus clientes. Cualquier computador puede ser un servidor(carísimos o baratos), incluso los **clusters de computadores** (muchos ordenadores juntos que se perciben como uno solo).

Nota ¿Puede un sistema empotrado ser un servidor?

La respuesta es ¡sí!. Un sistema empotrado, como una Raspberry Pi, pueden hacer las veces de servidor. También un ordenador de sobremesa puede serlo. La diferencia entre un ordenador hecho para un propósito específico y uno que es de uso general, es que las especificaciones y la arquitectura están especialmente diseñadas para esa tarea. Por poner un ejemplo, un servidor que sirva para hacer copias de seguridad, como un NAS, tendrá menos memoria y CPU que un PC de escritorio porque no le hacen falta para nada.

Tipos de servidores

- **Servidor web**: almacena el código y el multimedia de una página y lo distribuye a quien la visite.
- **Servidor de archivos**: Acceso a archivos almacenados o accesibles por él (como una interfaz de una base de datos).
- **Servidor de base de datos**: Archiva información de una forma específica en una base de datos.
- **Servidor de comercio electr.**: Procesa compras/ventas. Valida al cliente y genera el pedido.
- **Servidor de correo electr.**: Operaciones relacionadas con email (enviar, recibir, archivar...).
- **Servidor DHCP**: Otorga dinámicamente una IP a cada dispositivo que se conecte a una red.
- **Servidor DNS**: Devuelve la IP asociada a una url.
- **Servidor de impresión**: Controla impresora/s y ofrece servicios de impresión.

S.I. según la arquitectura de servicio

- **Aislado**: **Cada componente** del sistema (como aplicaciones) **funciona independientemente** sin depender de otros componentes internos ni externos. Cada sistema aislado tiene su propia base de datos, servidor y lógica de negocio.
- **Cliente/servidor**: Los **clientes** (como aplicaciones o usuarios) **solicitan recursos** a través de una conexión a un **servidor** que maneja las peticiones, **proporciona respuestas** y acceso a la información almacenada en su base de datos.
  - Puede haber un servidor que reparta la carga en otros servidores más sencillos que tienen bases de datos, por ejemplo.
- **Cliente-cola-cliente**: Varios clientes **comparten una misma infraestructura de servidor**, pero **operan independientemente** entre sí. Cada cliente tiene acceso a la base de datos y puede realizar solicitudes sin depender de los demás. Las solicitudes se encolan y distribuyen en los clientes por un servidor.

1.2 Fundamentos de Ingeniería de Servidores

Un servidor, como todo sistema informático, tiene elementos físicos (hardware), lógicos (software) y humanos (peopleware). Pero los elementos más importantes para él son los siguientes:

1.2.1. Prestaciones

Definición 1.2.1 Performance

Es la medida de la **velocidad** con la que **se realiza** determinada **carga** (*load/workload*) de trabajo

Hay dos medidas fundamentales en las prestaciones de un servidor:

1. **Tiempo de respuesta** (*response time*) o **latencia** (*latency*): Tiempo desde que se solicita una tarea hasta que se finaliza. Por ejemplo, la ejecución de un programa.
2. **Productividad** (*throughtput*) o **ancho de banda** (*bandwidth*): Cantidad de trabajo por unidad de tiempo. Por ejemplo, programas/hora.

Nota Sobre la productividad y el tiempo de respuesta

Aunque yo solicite a `ugr.es` ver su página y tarde 1 segundo, el servidor puede que tenga una productividad de 1000 solicitudes/seg. Depende del lado del servidor. La productividad se estanca cuando llegan más tareas de las que puedo gestionar por unidad de tiempo. Ahí se quedará en nuestro máximo. Por ejemplo, si tenemos un servidor muy potente, lo mismo llegamos hasta 1200 tareas por segundo, pero nunca subiremos de ahí porque el pobre no puede más. Sin embargo, el tiempo de respuesta es siempre creciente, puesto que cuanta más gente haya pidiendo un servicio, más cola hay para solicitarlo y más tiempo hay que esperar para obtenerlo.

Para mejorar nuestras prestaciones, podemos cambiar o ajustar el **hardware**, el **Sistema operativo**y las **aplicaciones** según nuestras necesidades.

1.2.2. Fiabilidad

Definición 1.2.2 Fiabilidad/Reliability

Probabilidad de que el servidor funcione correctamente. No puede tener fallos ni físicos (componentes estropeados o respuestas incorrectas) ni software (por programación). Se suele medir en fallos/segundo (FIT) o tiempo medio hasta el fallo (MTTF/MTBF).

Para mejorarla podemos utilizar **sumas de comparación** de bits (*checksums*) para **detectar errores** o comprobar paquetes, **sistemas de alimentación ininterrumpida** (SAI/*UPS*) para **evitar cortes de luz** que apaguen nuestro sistema o **sistemas redundantes de discos** (RAID) (o de alimentación, de red, etc.) que **detecten fallos de componentes y reaccionen** ante ellos.

1.2.3. Disponibilidad

Definición 1.2.3 Disponibilidad/Availability

% de tiempo en el que el servidor está operativo (responde a lo que le piden), sin necesidad de que sean correctas esas respuestas.

Definición 1.2.4 Tiempo de inactividad/Downtime

Cantidad del tiempo en el que el servidor no está disponible.

Hay dos tipos de inactividad

1. **Planificada**: Para mantenimiento o actualización del servidor.
2. **No planificada**: Debido a ataques o fallos del sistema.

Podemos mejorarla reemplazando componentes en caliente (hot-plugging o hot-swapping) o usar cualquier medida que aumente la fiabilidad.

Nota Fiablidad vs Disponibilidad

La diferencia entre Fiable y Disponible es la siguiente:
Imaginemos que nuestro ordenador explota - NO es fiable NI disponible.
Imaginemos que nuestro ordenador está encendido pero devuelve 2+2=9 - NO es fiable pero SI disponible.
Imaginemos que nuestro ordenador está encendido y además devuelve 2+2=4 - SI es fiable Y disponible.

1.2.4. Seguridad

Características

1. **Confidencialidad**: Evitar individuos no autorizados.
2. **Integridad**: Evitar corrupción o alteración maliciosa de datos.
3. **Anti-ataques**: Evitar impedimento al acceso de recursos.

Para ello podemos incluir antivirus, firewalls, encriptación de datos y otras medidas que veremos más tarde en el temario.

1.2.5. Mantenimiento

Definición 1.2.5 Mantenimiento

Acciones para prolongar el funcionamiento correcto del sistema.

Esta acciones incluyen, entre otros: Sistemas operativos actualizados, copias de seguridad y automatización de tareas.

1.2.6. Extensibilidad/Expansibilidad

Definición 1.2.6 Extensible/Expansible

Que puedes aumentar sus recursos.

Esto se puede conseguir teniendo conectores libres, racks de servidores, SO modulares o interfaces de E/S estándar. Cualquier mejora que permita **escalabilidad, aumentaría la expansibilidad**.

1.2.7. Escalabilidad

Definición 1.2.7 Escalabilidad

Poder **aumentar el tamaño de un sistema** (el número de usuarios que pueden acceder a él) **sin afectar a su eficacia o calidad**. Por ejemplo, si nos llegan 10000 peticiones a nuestro servidor, podemos dar dinámicamente más recursos (más servidores chiquititos) para que se pueda mantener el nivel de calidad de servicio.

Esto se logra con servidores modulares (*clusters*), arquitecturas por capas, programación paralela, etc. **Todos** los sistemas **escalables son extensibles** pero **no al revés**.

Nota **Expansible vs Escalable**

En resumen: Expansible es que "puedo ponerle cosas" escalable es que funciona igual independientemente de cuántos usuarios pidan servicio.
Un ejemplo de expansible pero no escalable sería añadir dos fuentes de alimentación, porque eso no afecta a la escalabilidad, sino a la disponibilidad.

1.2.8. Coste

Es el presupuesto que tienes para tu servidor, que no es infinito. Puedes reducirlo con código abierto o reduciendo costes de la electricidad. Todos lo selementos de un sistema informático (hardware, software, peopleware) cuestan dinero y hay que tenerlos en cuenta cuando lo creamos.

1.3 Introducción a la comparación de características entre servidores

1.3.1. Motivación: comparación de prestaciones

Queremos saber cuál es el más rápido para ejecutar un conjunto de programas. Cuando comparamos, hay dos formas de decirlo: cuántas **veces es más rápido** y qué **tanto por ciento de mejora** aporta.

Nota **% vs xVeces**

*Cuando decimos que un ordenador es x1 veces más rápido que otro, es **igual**. Esto es porque, si multiplicamos x1 las características, nos sale lo mismo.*
*Ahora, si decimos que un ordenador es **100 % más rápido** significa que es el **doble** de rápido que el anterior. Esto se debe a que aumenta un 100 % las capacidades, como si tuvieramos dos ordenadores del 100 % en uno solo.*

1.3.2. Tiempos de ejecución mayores/menores

Fórmula 1.3.1 xVeces

Si T_a= tiempo de ejecución de A y lo mismo para T_b.

$T_a/T_b = x$, T_a es xVeces T_b

Ejemplo: T_a = 10 seg, T_b = 5 seg, 10/5= 2, T_a es 2 veces T_b

Del mismo modo 5/10 =0.5, T_b es 0.5 veces T_a.

Fórmula 1.3.2 Cambio relativo

$(x\text{Veces} - 1) \cdot 100 = \%$ de cambio.

Ejemplo: Con los tiempos de antes salía A 2veces B, pues $(2 - 1) \cdot 100 = 100\,\%$ más rápido que B.

1.3.3. Ganancia

Fórmula 1.3.3 Ganancia/Speedup de A respecto a B

Si tenemos un programa A más rápido que B y queremos saber cuánto hemos ganado respecto a nuestra situación anterior:

$$S_B(A) = \frac{V_A}{V_B} = \frac{\frac{Trabajo}{T_A}}{\frac{Trabajo}{T_B}} = \frac{\frac{1}{T_A}}{\frac{1}{T_B}} = \frac{T_B}{T_A}$$

Para obtener el % de cambio

$$(S_B(A) - 1) \cdot 100$$

Nota ¿20 % más rápida significa que la otra es 20 % más lenta?

Un error muy común del examen es decir: "Ah, pues si A es 40 % más rápida que B, entonces B es 40 % más lenta que A".
***¡NO!**, los % son relativos a cada máquina. Pongo un ejemplo:*

T_A *= 36 seg,* T_B *= 45 seg.*

$S_B(A) = \frac{45}{36} = 1{,}25$ *y* $(1{,}25 - 1) \cdot 100 = 25\,\%$ *más rápida que B.*

$S_A(B) = \frac{36}{45} = 0{,}8$ *y* $(0{,}8 - 1) \cdot 100 = -20\,\%$*. 20 % más lenta que A.*

Como ves, salen números distintos. Recuerda siempre ***hacer ambos en el examen*** *para que no te pillen los profes.*

1.3.4. Relación prestaciones/coste

Imaginemos que queremos saber si mejorar a un componente más caro pero más rápido nos renta, utilizamos la relación entre las prestaciones y el coste.

Fórmula 1.3.4 Relación prestaciones/coste

La unidad en la que está el resultado de esta fórmula es s^{-1}/€

$$\frac{Prestaciones_A}{Coste_A} = \frac{v_A}{Coste_A} = \frac{\frac{1}{T_A}}{Coste_A}$$

Si usamos esta fórmula, tenemos que buscar **el número más grande**, puesto que es el que **mayor proporción coste/prestaciones** tiene.

Fórmula 1.3.5 Cuanto mayor/menor es la relación

El resultado de esta fracción es cuántas veces mayor es para nuestro programa A. Si queremos el %, necesitamos restarle 1 y multiplicarlo por 100.

$$\frac{\frac{Prestaciones_A}{Coste_A}}{\frac{Prestaciones_B}{Coste_B}}$$

1.4 Ley de Amdahl

Terminología

- T_o: **tiempo original** en ejecutar cierto proceso.
- k: **cuántas veces mejoramos** el componente que nos interesa. (k-1)$\cdot 100$ % más rápido
- f: **Fracción del tiempo** que se **usa el componente**. Si es f=1, se usa todo el tiempo.
- $1-f$: **Fracción del tiempo** que **NO** se **usa** el componente.
- T_m: **Tiempo mejorado** después de mejorar el componente.

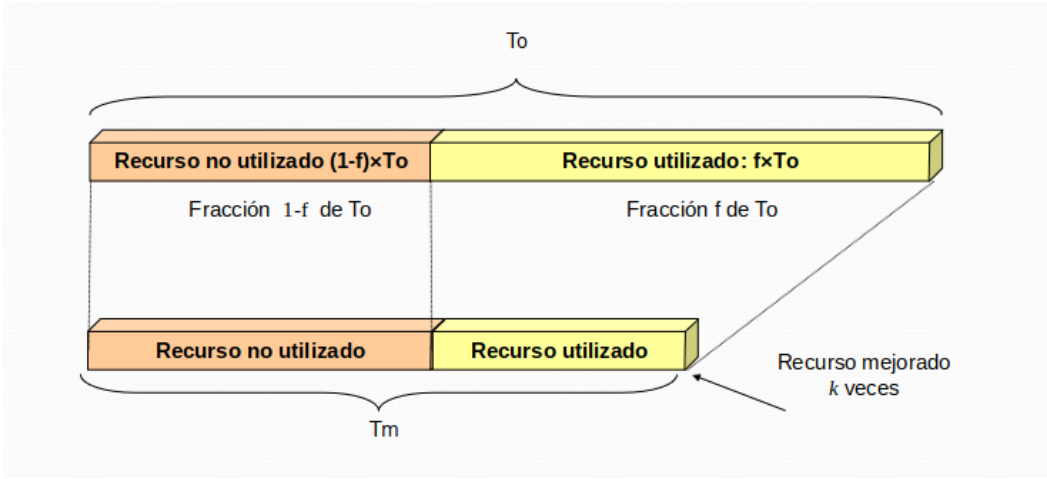

Figura 1.1: Tiempo original (T_o) vs tiempo mejorado (T_m)

Fórmula 1.4.1 Ley de Amdahl

Recordemos que la ganancia era tiempo B/tiempo A. Sabiendo esto y cómo llamamos a cada trocito de nuestro programa antes y después de mejorarlo, sale la siguiente fórmula:

$$S = \frac{T_B}{T_A} = \frac{T_o}{T_m} = \frac{T_o}{(1-f)\cdot T_o + \frac{f\cdot T_o}{k}} = \frac{1}{1-f+\frac{f}{k}}$$

Dado que $(1-f)\cdot T_o$ no varía y $f\cdot T_o$ lo mejoramos k veces (lo dividimos entre k), T_m es la

suma de esos dos.

Nota **Casos particulares**

- *Si $f = 0$, significa que ninguna fracción del programa utilizamos el componente que vamos a mejorar, así que no ganaremos nada.*
- *Si $f = 1$, significa que el sistema mejora tanto como el componente, puesto que lo usamos en la totalidad del tiempo.*
- *Si $k \to \infty$, significa que estamos mejorando TODO LO POSIBLE el componente. Tanto tanto, que es casi como si su parte **se ejecutara instantáneamente**. Así que mejoramos tanto como pudieramos, **el máximo posible**, que es $\frac{1}{1-f}$.*

Las veces que mejoramos el componente (k) a**fectan más cuanto más tiempo utilicemos el componente** en nuestro programa. Evidentemente, si solo lo empleamos unos microsegundos, la ganancia será despreciable y deberíamos centrar nuestros esfuerzos en otros componentes más solicitados.

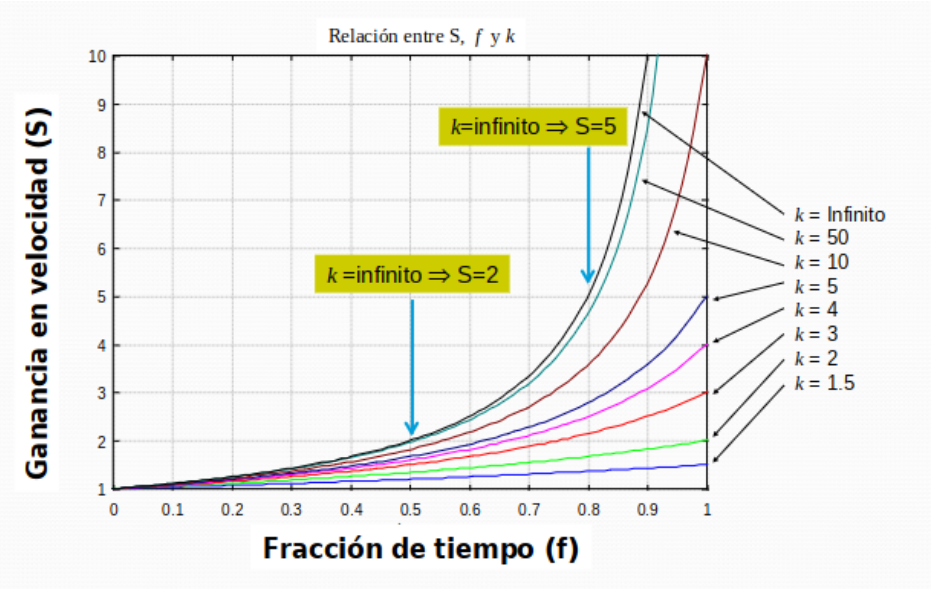

Figura 1.2: Relación entre ganancia y fracción del tiempo que usa el componente

Fórmula 1.4.2 Ley de Amdahl con muchos componentes

Es la misma fórmula, pero **sumas las f afectadas** para calcular el **tiempo que no se toca** (1-f) y sumas **los componentes** que mejoras **con sus respectivas** k:

$$\frac{1}{(1-\sum_{i=1}^{n} f_i) + \sum_{i=1}^{n} \frac{f_i}{k_i}}$$

En otras palabras: "Uno menos todas las fracciones de tiempo que mejoro + todas las fracciones que mejoro partidas de su respectiva mejora"

Nota Dónde encontrar ejercicios de Ley de Amdahl

Hay un montón de ejercicios resueltos en wuolah y son iguales que años pasados pero, si te quedas con ganas, también hay más en la asignatura de arquitectura de computadores de segundo año. ¡Mucho ánimo!