Diseño y Arquitectura de Servicios Escalables

Unidad 3
Escalabilidad

Índice

Bibliografía

[DG04] Jeffrey Dean, Sanjay Ghemawat: “MapReduce: Simplified Data Processing on Large Clusters”, OSDI 2004: 137-150.

[GL02] Seth Gilbert, Nancy A. Lynch: “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”. *SIGACT News* 33(2): 51-59, 2002.

[Hil90] Mark D. Hill: “What is Scalability?”, *ACM SIGARCH Computer Architecture News*, 18(4):18–21, December 1990.

[JW00] Prasad Jogalekar and Murray Woodside: “Evaluating the Scalability of Distributed Systems”, *IEEE Trans. Paral. Distrib. Syst.* 11(6):589-603, June 2000.

[Neu94] B. Clifford Neuman: “Scale in Distributed Systems”. In T. Casavant and M. Singhal, (eds.), *Readings in Distributed Computing Systems*, pp. 463–489. IEEE-CS Press, 1994.

[Sto86] Michael Stonebraker: “The Case for Shared Nothing”. *IEEE Database Eng. Bul.,* 9(1):4-9, March 1986.

Índice

1. Definición de escalabilidad

- De manera informal, escalabilidad es...
  - La capacidad para que un sistema...
    - mantenga su eficiencia...
    - (y calidad de servicio...)
    - a medida que aumente su uso de recursos de cómputo.
- Esta definición es válida en sistemas multiprocesador orientados a computación intensiva, donde...
    - Solo se considera una aplicación en cada momento.
    - Solo existe un “usuario”.
    - Las restricciones inherentes a una arquitectura multiprocesador limitan la disponibilidad de recursos.

1. Definición de escalabilidad

- Pero no es una definición aplicable a sistemas distribuidos [JW00], en los que:
  - Podrá haber múltiples aplicaciones compitiendo por los recursos.
  - Habrá múltiples usuarios interesados en cada aplicación.
  - Habrá que considerar el uso de la red.
    - Es decir, habrá múltiples tipos de recurso (no sólo procesadores).
  - Es posible que los ordenadores pertenezcan a múltiples organizaciones.

1. Definición de escalabilidad

- Según Mark Hill [Hil90], al empezar la década de los '90 no había ninguna definición formal de escalabilidad:
  - "*I examined aspects of scalability, but did not find a useful, rigorous definition of it. Without such a definition, I assert that calling a system "scalable" is about as useful as calling it "modern".*"

1. Definición de escalabilidad

- Según Neuman [Neu94]:
  - "*A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.*"
- Sigue siendo una definición informal.
- Pero se complementó especificando tres dimensiones "escalables".

1. Definición de escalabilidad

- En [JW00] se dio una primera definición formal:
  - Se asume algún factor de escala “k”.
    - Ejemplos: número de usuarios, número de nodos, tamaño de la base de datos...
  - La métrica de escalabilidad se basa en la productividad:
    - Si la productividad se mantiene a medida que la escala (valor de “k”) se incrementa, el sistema se considerará escalable.
    - Pero no se fija exclusivamente en la productividad, sino que también considera la calidad de servicio.

1. Definición de escalabilidad

- En [JW00] se dio una primera definición formal (cont.):
  - F(k) = T(k)*f(k)/C(k), donde:
    - F(k): productividad con una escala “k”.
    - T(k): rendimiento en respuestas/segundo (escala “k”).
    - f(k): beneficio promedio de cada respuesta, en función de la calidad de servicio (QoS) ofrecida con una escala “k”.
    - C(k): coste, expresado como coste por segundo, para poderlo componer con T(k).
  - A partir de F(k), se define la escalabilidad entre dos factores k1 y k2 (“*Scal(k1,k2)*”) como:
    - Scal(k1,k2) = F(k2)/F(k1)
    - El sistema se considerará escalable si Scal(k1,k2) se aproxima a la unidad.
    - [JW00] recomienda que Scal(k1,k2)>0.8 para que un sistema se considere escalable.

1. Definición de escalabilidad

- Volviendo a la interpretación de “k”:
  - En un sistema distribuido habrá múltiples factores de escala.
  - ¿Cómo considerarlos todos?
    - Separando entre “x(k)” (variables de escalabilidad) e “y(k)” (habilitadores de escalabilidad).
    - Ejemplos de habilitadores: estrategias de asignación de procesadores a procesos, prioridades, estrategias de replicación, uso de hilos, configuración del middleware, ancho de banda disponible...

1. Definición de escalabilidad

- Ejemplo: Un sistema de gestión de datos distribuido:
  - Número de usuarios activos (factor básico, k). x0(k)=k
  - Tamaño de la base de datos, en registros (x1): x1(k)=10000*log(k)
  - Número de nodos (x2): x2(k)=k/100
- Es decir, se definirá alguna relación entre los diferentes factores o variables a partir de un factor básico.
- Se utilizarán las diferentes x(k) para construir las expresiones del coste (C(k)) y, en algunos casos, también del beneficio (f(k)).
  - Depende de cada ejemplo particular.
  - Relación con teoría de colas.
  - Ver ejemplo en la siguiente sección.

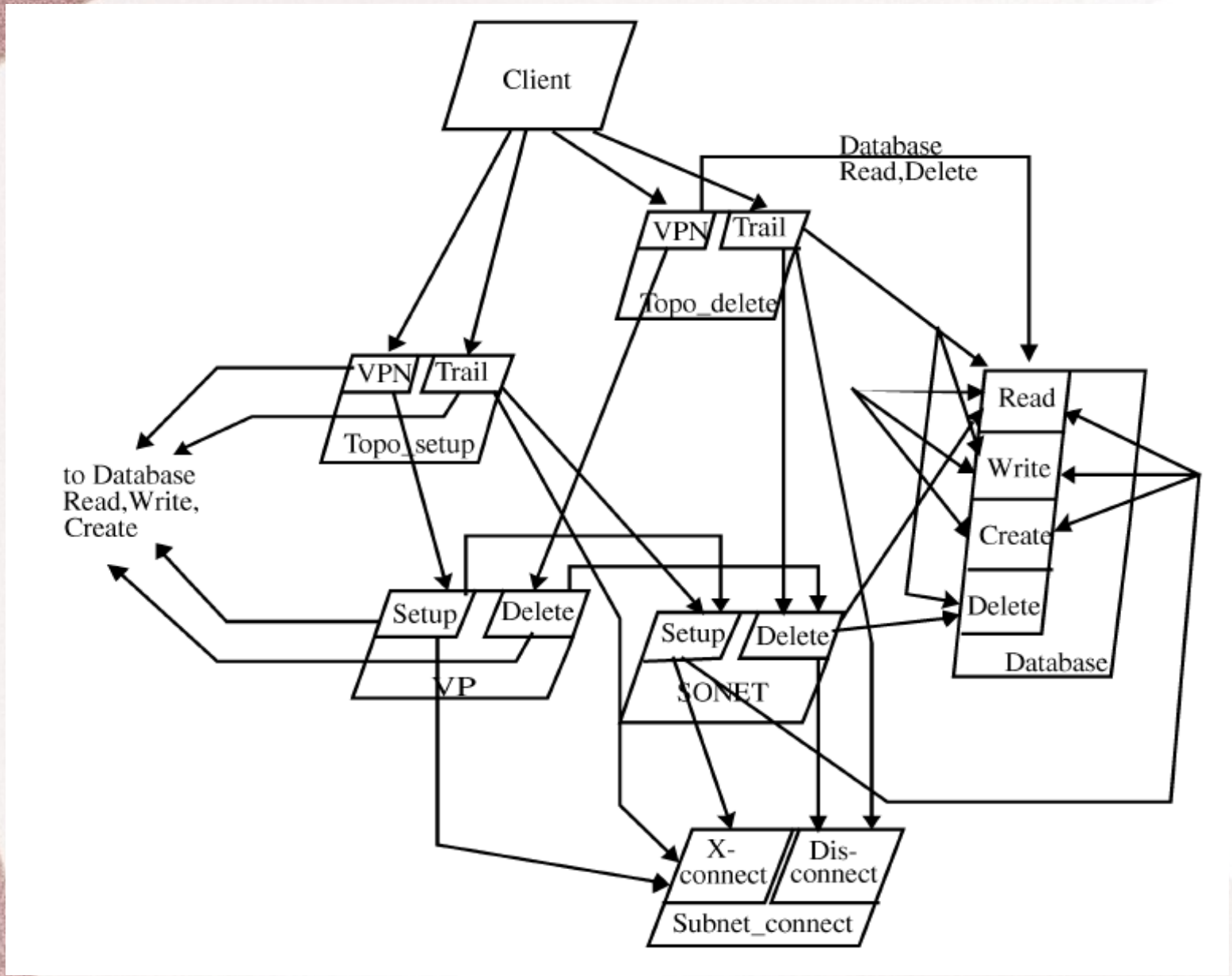
1.1. Ejemplo de aplicación

- En [JW00] se presenta un ejemplo de un sistema de gestión de conexiones a una VPN para vídeoconferencia.
  - Se distinguen tres niveles en su arquitectura software:
    - Nivel de topología (VPN): Conecta los “end-points” especificados por los usuarios, generando una VPN entre ellos.
    - Nivel de camino virtual (VP): Establece las rutas dentro de la VPN por las que se transmitirán los paquetes.
    - Nivel SONET: Es el que gestiona la red física. Topología de anillo, en este sistema.

1.1. Ejemplo de aplicación

- Se distinguen las siguientes seis tareas:
  - TopoSetup, TopoDelete: Dos tareas del nivel de topología. Establecen las conexiones dentro de la VPN.
  - VP: Tarea que establece los caminos virtuales (VPs) dentro del nivel intermedio.
  - SubnetConnect: Gestiona los elementos públicos del nivel SONET.
  - SONET: Gestiona los recursos internos del nivel SONET.
  - Database: Gestiona los datos manejados por todas las demás tareas.

1.1. Ejemplo de aplicación



1.1. Ejemplo de aplicación

- La información gestionada por el componente “Database” es accedida por todos los demás componentes.
- Es el “cuello de botella” de esta aplicación distribuida.
- Será el componente con mayor relevancia en el análisis de escalabilidad.
- Por tanto, el factor de escala básico “k” será el número de réplicas de la base de datos.

1.1. Ejemplo de aplicación

- La configuración básica estaba formada por:
  - Un nodo para cada una de las cinco tareas no relacionadas con la base de datos. 5 nodos.
  - Un nodo por cada réplica de la base de datos. “k” nodos, con “k” mayor o igual a uno.
  - Los costes son:
    - 100000$ para la configuración básica (seis máquinas y las licencias correspondientes).
    - 5000$ por cada réplica adicional.
    - Esto implica un coste por segundo igual a:

      C*(1+0.05k)$
    - Siendo C=0.001 en caso de que la vida útil de estos componentes ronde los 3 años.

1.1. Ejemplo de aplicación

- A partir de esta información, se calculan los límites para la escalabilidad, a partir de otras fórmulas descritas en [JW00]:
  - La configuración base permite atender a 23 clientes, con una productividad F(1)=0.0195 $/seg.
    - Mide el beneficio económico obtenido por cada unidad temporal.
  - Se computan los siguientes valores base:
    - Demanda total: D=14.44+22.11k seg.
    - Demanda media: D_{avg}=(14.44+22.11k)/(k+5) seg.
    - Demanda máxima: D_{max}=35.08 seg.
    - Tiempo de reacción (think time): Z=600 seg.
    - Tiempo de respuesta (N=número de clientes):

$$R=D+(N-1)D_{avg}/(1+(Z/D))$$

1.1. Ejemplo de aplicación

- Se realiza el estudio de escalabilidad, con los siguientes resultados:

Scalability Analysis Results for the Connection Management System

Scale Factor	Productivity (Optimized) (sec^{-1} per unit cost) x 1e-2	Scalability metric value (Optimized)	Throughput (operations per hour)	Normalized* Response Time	Database CPU Utilization		System Cost (units)
					Total	Due to transaction overheads	
1	1.95485	1.0	95	0.3017	92.59	--	1.05
2	2.02031	1.0335	108.62	0.3645	86.86	67.85	1.1
3	1.90128	0.9726	111.18	0.4126	82.43	69.45	1.15
4	1.75662	0.8986	112.01	0.4761	79.77	69.97	1.2
5	1.61546	0.8264	112.26	0.5449	77.98	70.12	1.25
6	1.4861	0.7602	110.95	0.5953	75.77	69.30	1.3
7	1.36948	0.7006	110.94	0.6675	74.84	69.30	1.35

*The normalized response time is the mean response-time divided by the target of 15 min.

Índice

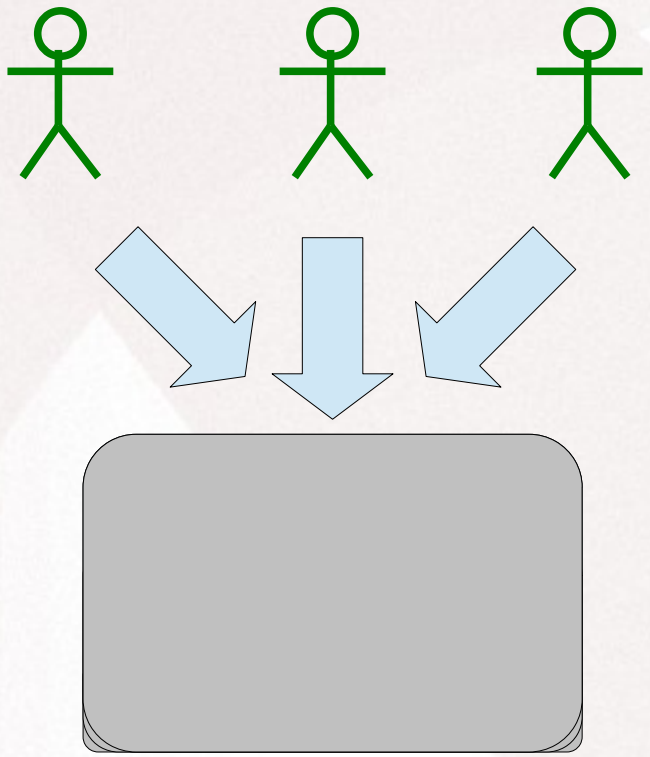
2. Dimensiones de escalabilidad

- Las tres dimensiones de la escalabilidad son [Neu94]:
  - *Escalabilidad de tamaño*: Cuando el número de usuarios que puedan ser atendidos por un servicio distribuido crezca linealmente con el número de nodos del sistema.
  - *Escalabilidad geográfica*: Cuando el sistema tolere los problemas que comporta una mayor distancia entre sus nodos (mayor retardo y menor fiabilidad en la comunicación).
  - *Escalabilidad administrativa*: Cuando el sistema admita que múltiples organizaciones colaboren en la administración de sus nodos.

2. Dimensiones de escalabilidad

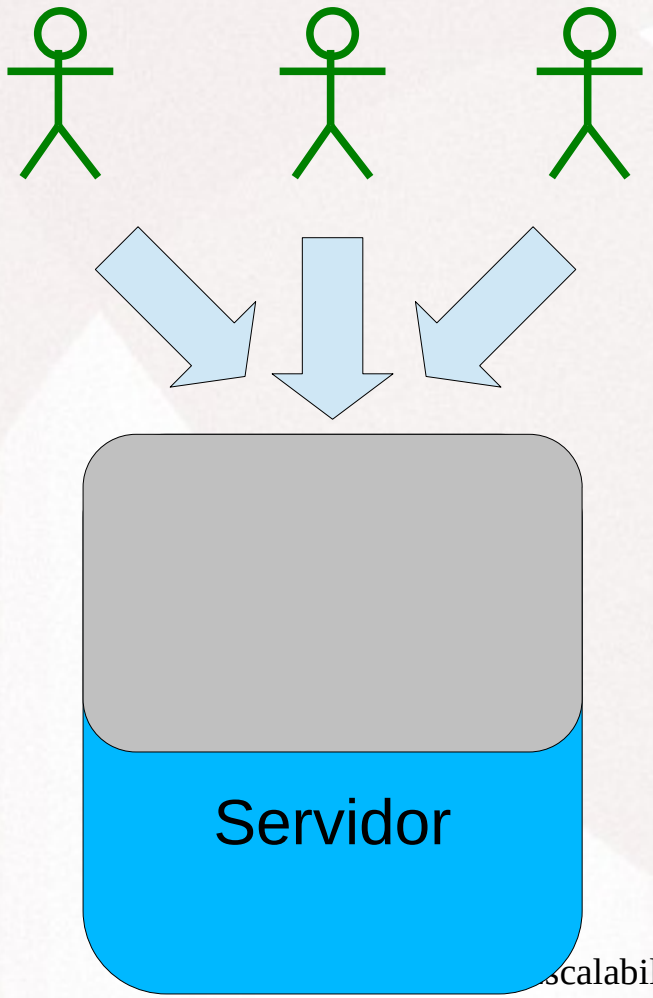
- Hoy en día también se distingue entre:
  - Escalabilidad vertical (“scale up”): Incremento en la capacidad de un único nodo.
    - Reemplazando sus componentes “hardware”.
    - Puede darse en cualquier sistema informático (no necesariamente distribuido).
  - Escalabilidad horizontal (“scale out”): Incremento en el número de nodos de un sistema.
    - Similar a la escalabilidad de tamaño identificada por Neuman.
    - Es la que importa en un sistema distribuido.

2.1. Escalabilidad vertical



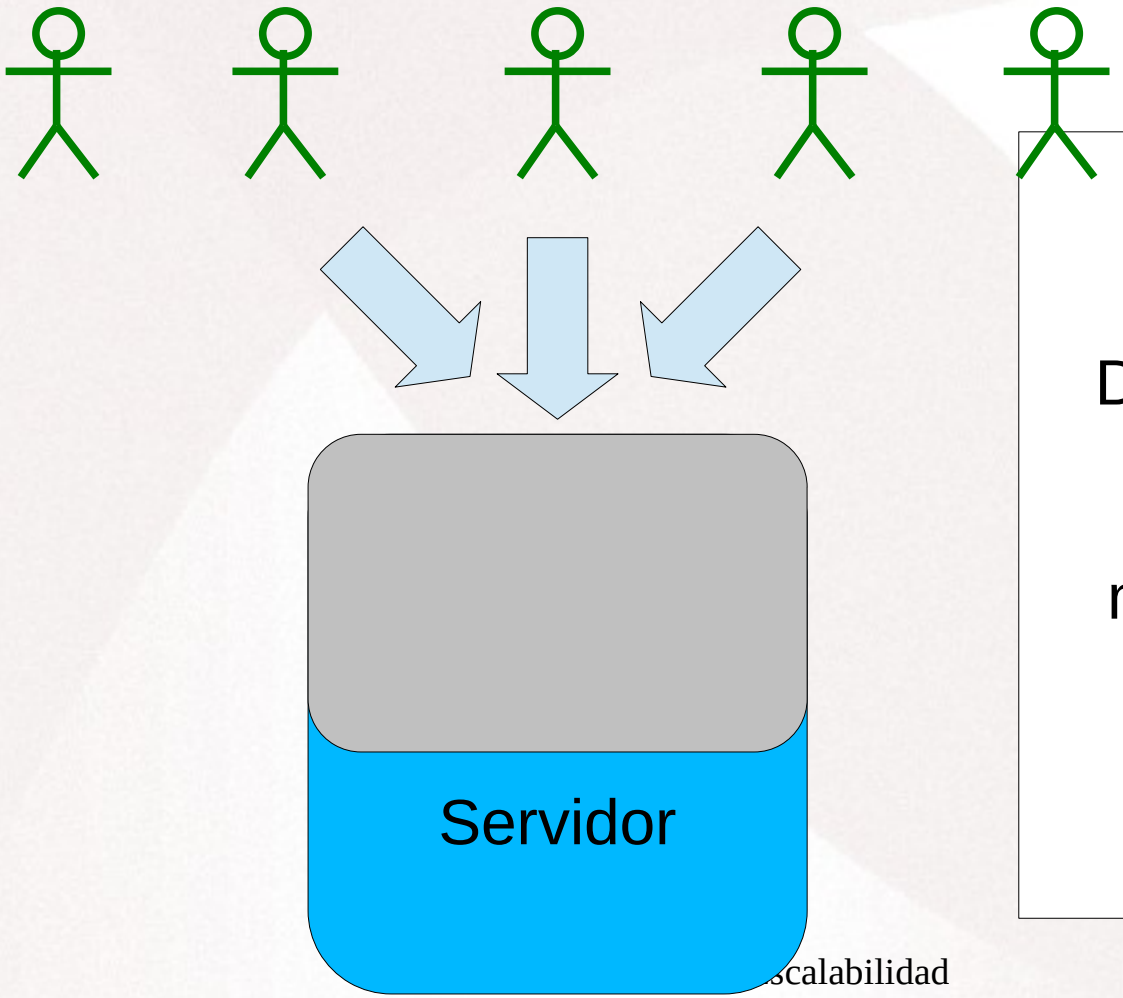
Un servidor puede llegar a saturarse con una carga determinada.

2.1. Escalabilidad vertical



La escalabilidad vertical consiste en incrementar la capacidad del nodo.

2.1. Escalabilidad vertical

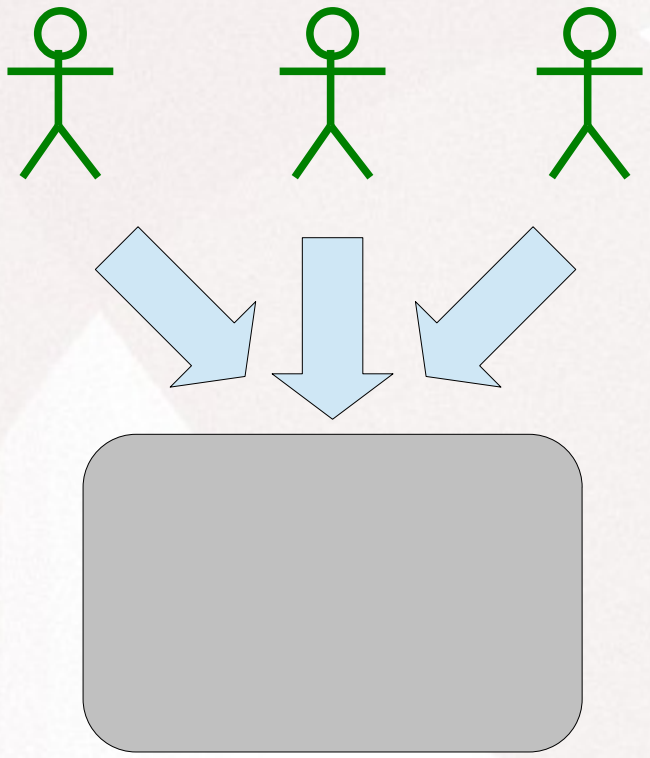


De esa manera se podrá atender a un mayor número de usuarios.

2.1. Escalabilidad vertical

- Pero la escalabilidad vertical tiene limitaciones severas:
  - Suele ser caro el reemplazo de componentes.
  - Requiere parar y reconfigurar la máquina para aprovechar esas mejoras.
  - No siempre se pueden encontrar mejores componentes.
  - Llega un punto en que realizar más ampliaciones resulta inviable.

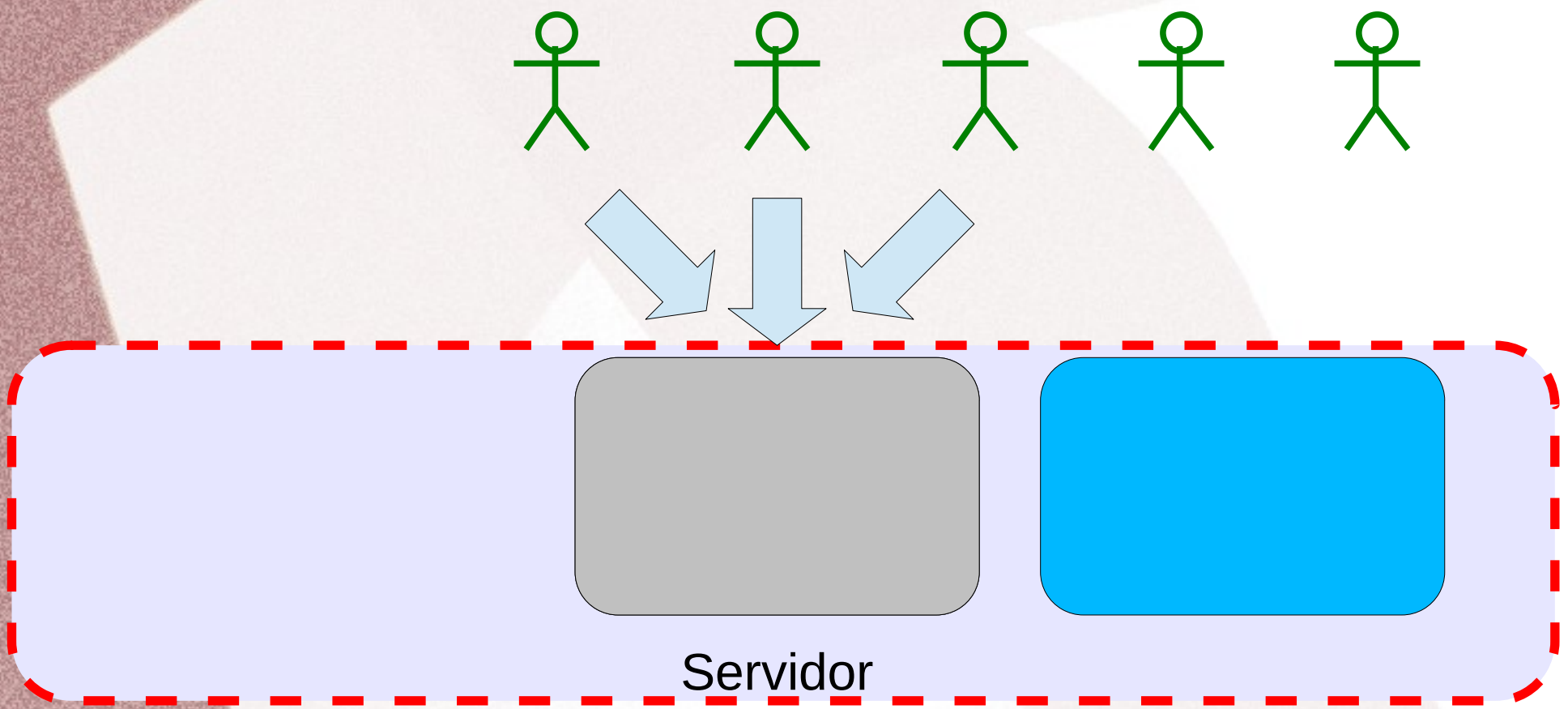
2.2. Escalabilidad horizontal



Servidor

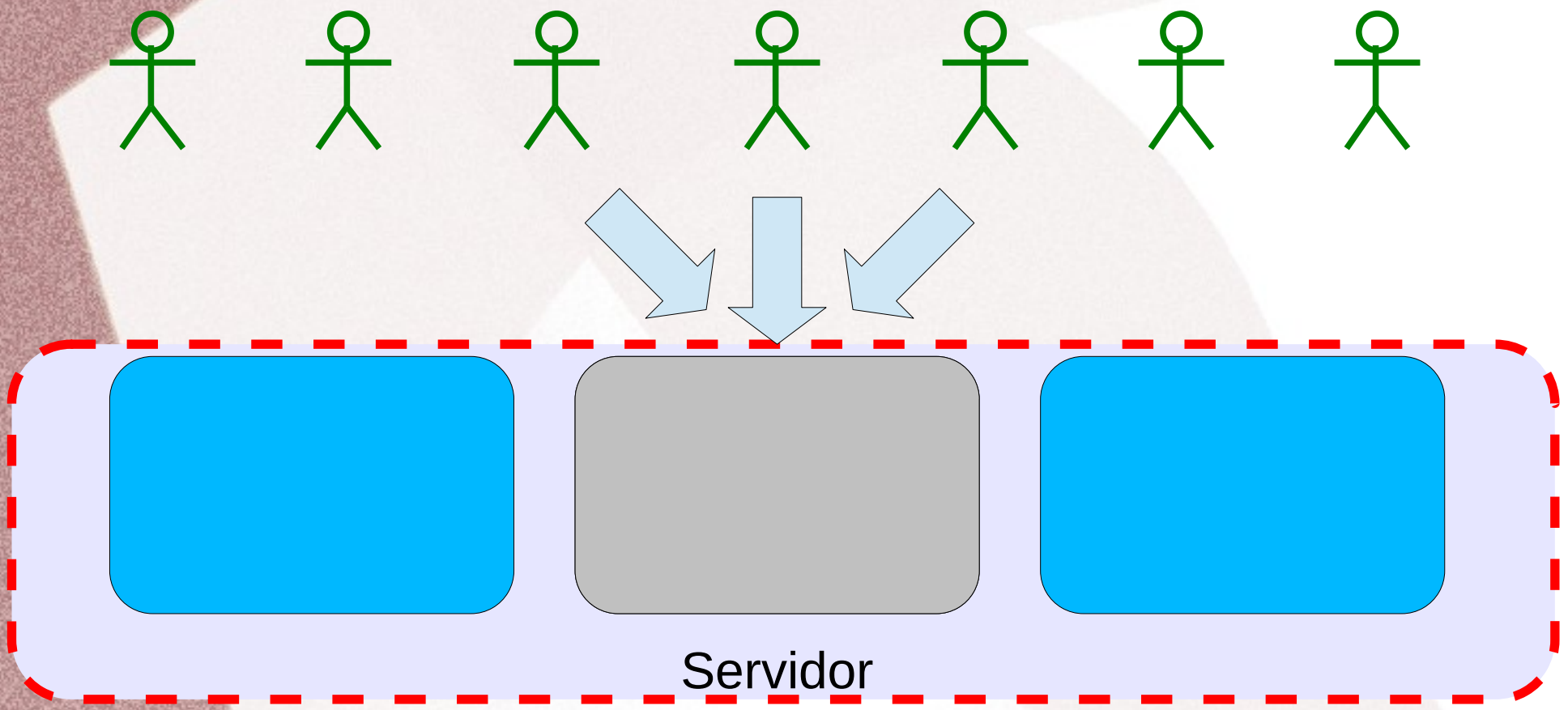
Cuando un servidor se sature...

2.2. Escalabilidad horizontal



… se replica en otros nodos y así incrementa su capacidad de servicio.

2.2. Escalabilidad horizontal



Si la carga se siguiere incrementando, se añadirían más réplicas.

2.2. Escalabilidad horizontal

- La escalabilidad horizontal es más sencilla:
  - El coste de añadir nuevas máquinas suele ser bajo.
  - El esfuerzo de reconfiguración no es grande.
  - No se pierde disponibilidad de servicio con una reconfiguración cuidadosa.

Índice

3. Escalabilidad de tamaño

- Para obtener escalabilidad de tamaño, se debe utilizar:
  - División/reparto de tareas.
  - División/reparto de datos.
  - Replicación.
  - Caching.

3.1. División de tareas

- Debe evitarse la centralización de la actividad.
  - Implicaría que un nodo fuera el responsable de la toma de decisiones...
    - Esto no siempre será malo, a menos que implique bloqueos prolongados para los demás.
  - ... y de la ejecución de la mayor parte de las actividades.
    - Esto sí es problemático.
    - Ese nodo podría saturarse.

3.1. División de tareas

- Las tareas deben repartirse entre todos los nodos.
  - Pero debe hacerse de manera que no requiera sincronización adicional.
  - Para ello suele ser conveniente repartir también los datos.
  - Interesará delegar aquellas tareas que sea posible a los propios clientes.
  - Conviene utilizar algoritmos descentralizados.

3.2. División de datos

- Conviene repartir ("*sharding*") el conjunto de datos que gestione cada aplicación entre los diferentes nodos [Sto86]. Así:
  - Cada nodo es responsable de un subconjunto de los datos.
    - Resulta más fácil mantenerlos (tanto en RAM como en disco).
      - Conjunto menor en cada nodo.
    - Cada nodo recibirá una parte proporcional de peticiones.
  - Se incrementa el grado de concurrencia.
    - Para ello, debe evitarse la sincronización entre los nodos.

3.2. División de datos

- Reparto / “sharding”:
  - En base a cierta clave (conjunto de atributos primarios) se realiza el reparto.
  - Cada nodo obtiene un “fragmento” de la base.
    - Fragmento (“*shard*”): intervalo contiguo según la clave elegida.
  - Si la clave se elige de manera cuidadosa, una alta proporción de consultas sólo necesitarán acceder a un fragmento.
  - Mejor reparto del trabajo. Mayor concurrencia: cada nodo atiende una consulta/transacción diferente.

3.2. División de datos

- Interesaría un particionado perfecto.
  - Así cada operación sólo afectaría a un servidor.
  - Teóricamente posible, pero difícil de garantizar en la práctica.
  - Resultado: se utiliza replicación parcial.
    - Combinación de la división de los datos entre los nodos (mejora rendimiento) y la replicación de datos (incrementa su disponibilidad).
    - Aparece el compromiso entre consistencia, disponibilidad y particionado de la red (Teorema CAP [GL02]).

3.2. División de datos

- La división del conjunto de datos entre los diferentes nodos ha facilitado la base para el paradigma de programación Map-Reduce [DG04].
  - Asume accesos de solo lectura.
  - Una consulta se ejecuta en todos los nodos que mantengan algún fragmento de la base de datos.
  - Se ejecuta una subconsulta/filtrado (“map”) en cada nodo.
  - Ejecución concurrente de estas operaciones.
  - Después se integran los resultados parciales con el paso “reduce”, para generar un resultado final.

3.2. División de datos

- Otros ejemplos:
  - DNS: Proporciona un servicio de nombres distribuido de ámbito mundial.
    - Estructura jerárquica.
    - Particionado perfecto.
    - Reparto de responsabilidades claro.
    - Buena eficiencia con el uso de cachés.
      - Pero es un servicio donde los datos manejados no necesitan ser modificados con frecuencia.

3.3. Replicación

- Tanto los servidores como los datos manejados por éstos estarán replicados. Así:
  - Habrá múltiples copias de cada dato.
    - Se podrá equilibrar la carga entre todas ellas.
    - Podrán ser accedidas simultáneamente por múltiples procesos a la hora de realizar consultas.
    - Las peticiones podrán ser atendidas por la réplica más próxima (o la más descargada).

3.3. Replicación

- Esto garantiza escalabilidad lineal (y potencialmente infinita) para los accesos de lectura.
- ¿Qué ocurre con las modificaciones?
  - Implicarán retardos, pues deben ser propagadas y aplicadas en todas las réplicas.
  - El intervalo y el protocolo necesarios dependerán del modelo de consistencia proporcionado.

3.3. Replicación

- En las secciones 3.1 y 3.2 se indicó que había que minimizar la sincronización.
  - Eso obliga a adoptar consistencias relajadas.
  - La consistencia relajada es también consecuencia del teorema CAP [GL02], si queremos garantizar la disponibilidad de servicio cuando haya particiones en la red.

3.4. Caching

- *Caching*=Uso de cachés.
- Las cachés son un tipo particular de replicación, iniciada por el cliente:
  - Las cachés son mantenidas en los procesos clientes (localidad) o en servidores intermedios (mayor ratio de aciertos).
  - Se accede con menor frecuencia a los servidores originales.
    - De esta forma el servicio escala más fácilmente.

Índice

4. Escalabilidad de distancia

- Este tipo de escalabilidad aparece cuando...
  - ...la aplicación o servicio se “despliega” en nodos distantes entre sí.
    - Se incrementa el retardo de transmisión de los mensajes.
    - Se reduce la fiabilidad de esa transmisión.
    - Los algoritmos utilizados deben tener en cuenta esos “defectos”, compensándolos de alguna manera.

4. Escalabilidad de distancia

- ¿Cómo gestionar esos “defectos”?
  - Minimizando la necesidad de comunicación y sincronización entre componentes.
    - Algoritmos descentralizados.
    - Acciones locales siempre que sea posible.
    - Consistencia relajada.
      - Propagación diferida.
  - Ejemplos de servicios con este tipo de escalabilidad:
    - DNS: Estructura jerárquica, uso de cachés...
    - Sistemas P2P: DHT (estructurados y descentralizados).

4. Escalabilidad de distancia

- ¿Cómo gestionar esos “defectos”?
  - Otra solución complementaria consiste en utilizar comunicación asincrónica.
    - El emisor continúa sin preocuparse por obtener alguna respuesta.
    - Esto modifica el modelo de programación cliente-servidor tradicional, basado en RPC, donde se esperaba la respuesta del servidor.
    - También mejora la escalabilidad de tamaño.

4.1. Comunicación asincrónica

- Se dice que un mecanismo de comunicación basado en mensajes es asincrónico cuando:
  - El emisor no se bloquea al enviar un mensaje.
    - Puede continuar tan pronto como el mensaje llegue al middleware de comunicaciones del nodo local (emisor).
    - Poco importará la distancia que haya entre emisor y receptor.

4.1. Comunicación asincrónica

- Ejemplos de mecanismos de comunicación asincrónica:
  - 1) RPC asincrónica:
    - RPC sin respuesta.
    - No se especifican parámetros de salida ni se esperan resultados.
    - Se modifica el estado del servidor.
    - Los cambios generados serán observables en futuras invocaciones.
    - Comunicación uno-a-uno.

4.1. Comunicación asincrónica

- Ejemplos de mecanismos de comunicación asincrónica:
  - 2) Modelo “publish-subscribe”:
    - Dos roles diferentes: publicador y suscriptor.
    - Un middleware almacena los “eventos” publicados y los distribuye entre los suscriptores.
    - Comunicación muchos-a-muchos.

4.1. Comunicación asincrónica

- Ejemplos de mecanismos de comunicación asincrónica:
  - 3) Programación orientada a eventos:
    - JavaScript / NodeJS, por ejemplo.
    - El envío de un mensaje no bloquea al emisor.
    - La recepción de un mensaje es un evento más.
      - Queda pendiente en la cola de servicio.
      - Será procesado cuando corresponda.

Índice

5. Escalabilidad administrativa

- *Escalabilidad administrativa*: Cuando el sistema admita que múltiples organizaciones colaboren en la administración de sus nodos.
  - Pero cada organización tiene sus propios administradores.
  - Cada administrador decide qué políticas utilizar y sobre qué servicios de seguridad.
  - No será sencillo que una organización confíe en la gestión realizada en otras organizaciones.
    - Especialmente cuando cada organización se apoye en diferentes servicios de seguridad.

5. Escalabilidad administrativa

- Sistemas abiertos: Aquellos que utilicen mecanismos estándar, que faciliten la interoperabilidad.
- Puede que exista un middleware que integre los nodos de múltiples organizaciones.
  - Ejemplo: Globus Toolkit (en el campo de los sistemas Grid).

5. Escalabilidad administrativa

- En los sistemas cloud...
  - Todavía no existe ningún estándar que fuerce la integración de diferentes plataformas.
    - Cada plataforma puede tener objetivos diferentes, soportando interfaces distintas (pues los servicios a soportar también son distintos).
  - Cada proveedor IaaS es capaz de gestionar diferentes formatos de “máquina virtual”.
    - La portabilidad entre diferentes proveedores no debería ser excesivamente difícil.