Decisiones de Diseño

Sistemas Distribuidos

Contenido

- Latencia vs. throughput
- Consistencia, disponibilidad, tolerancia a particiones
- Servicios con estado o sin estado
- Diseños reactivos (iniciados por E/S)
- Localidad de datos
- Caso de estudio: Redes de distribución de contenidos

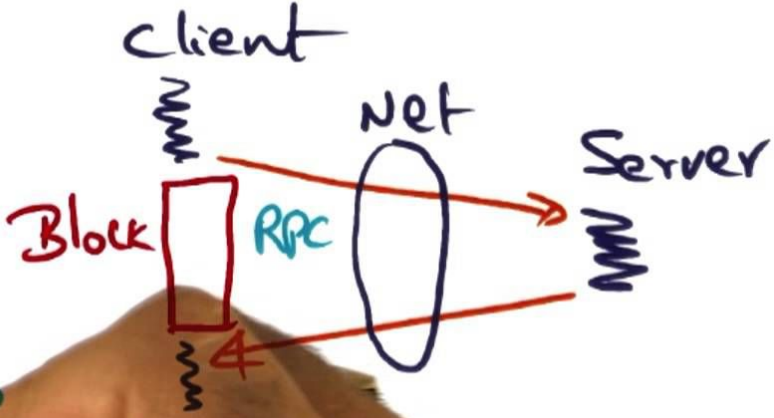
Design Tradeoff: Latencia vs. Throughput

Latency Vs. Throughput

Latency
- Elapsed time

Throughput
- Events per unit time
- Bandwidth: thruput measure

RPC performance



Latencia

- Mide el tiempo (máximo, 99 percentil, o promedio) que nos toma:
  - Completar un trabajo
  - En recibir la respuesta a una invocación remota
  - Transmitirse el primer byte de un conjunto de datos

Throughput (¿Rendimiento?)

- Mide:
  - Número de trabajos completados por segundo
  - Cantidad de invocaciones remotas atendidas por unidad de tiempo
  - Cantidad de datos transmitidos por unidad de tiempo

Latencia vs. throughput

- En un Sistema Distribuido, se debe analizar cuál de estos dos se debe optimizar
- Si no estoy seguro: Optimizar la latencia
- Ejemplo:
  - HDFS optimiza el throughput y no la latencia: puede transmitir grandes cantidades de datos de manera rápida, pero el primer registro de estos datos no llega muy rápido debido a que hay un overhead asociado con contactar el servidor de metadatos (Namenode)

Ejercicios

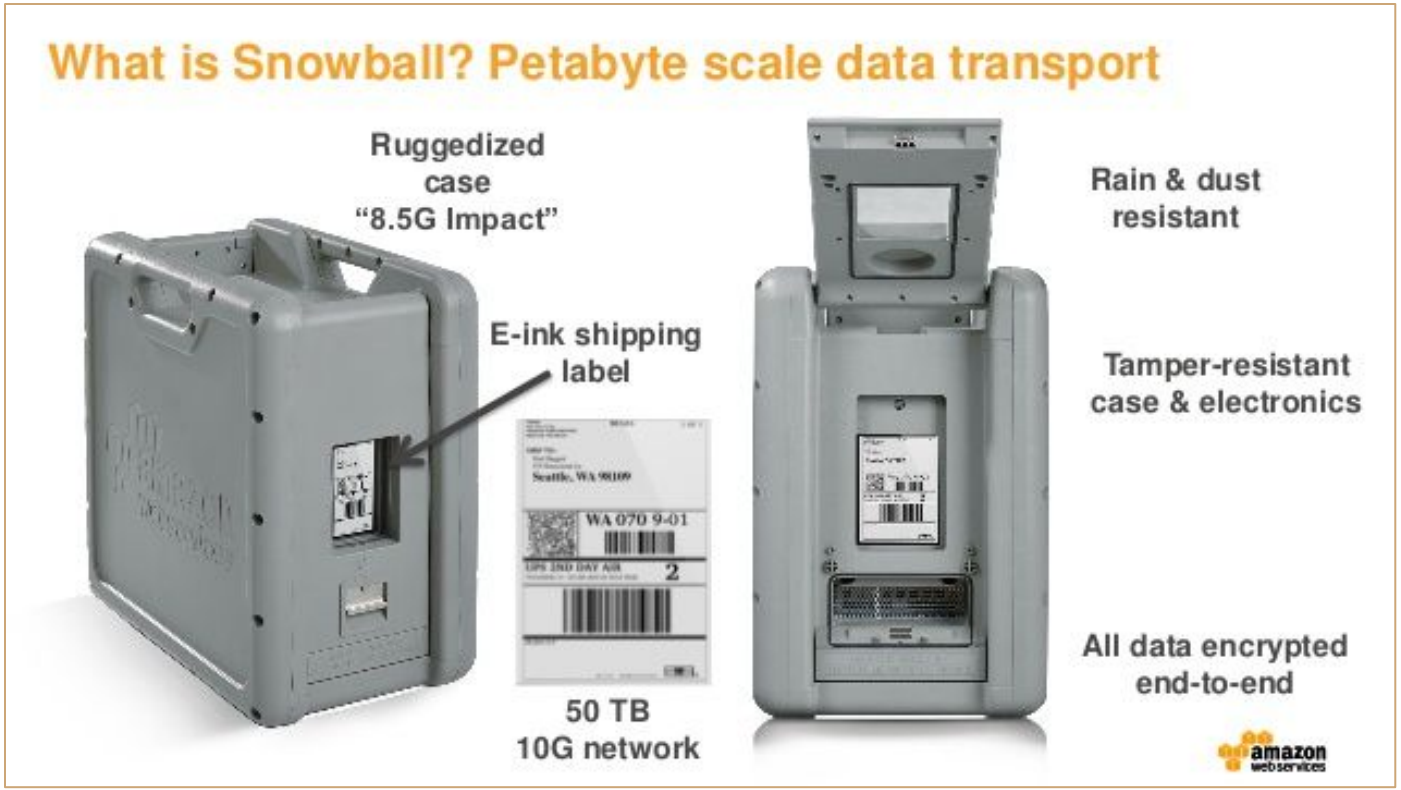
- ¿Qué se debe optimizar en los siguientes casos?
  - Sistema de procesamiento distribuido por lotes
  - Sistema de consultas ad-hoc distribuidas
  - Una base de datos de clave-valor

Ejercicio

¿Cuánto nos tardaríamos en enviar 1 PB de una empresa a otra si la conexión a Internet es de 20 Gigabits por segundo? (NOTA: El 1 PB contiene un dump de una base de datos muy grande, que ha sido comprimido para tener que enviar menos datos por el canal)

¿Cómo podemos hacer para mejorar la latencia del envío?

Ejemplo 1: AWS Snowball



Ejemplo 2: AWS Snowmobile

Consistencia y Teorema CAP

Design Tradeoff: Consistencia vs. Latencia

- Consistencia en SD requiere *consenso y acuerdo*
- Consenso y acuerdo → Complejidad y protocolos complejos → Mayor latencia
- No siempre hace falta 100% de consistencia
  - En esos casos, se puede mejorar latencia a costa de perder algo de consistencia

- **Ejercicio:** Liste tres ejemplos de SD que necesitan alta consistencia y tres ejemplos que puedan sacrificar consistencia para mejorar latencia

Ejemplos de SD que necesitan alta consistencia

- Sistema de facturación electrónica
- Juego de ajedrez distribuido
- Sistema de banca electrónica
- Sistema de compras por Internet
- En general: RDBMS
- Sistema de control de tráfico aéreo

Ejemplos de SD que **NO** necesitan alta consistencia

- Sistema de conteo de clicks de clientes Web (ad management)
- Caché replicada (si todas las réplicas no contienen el 100% de las mismos objetos, no importa mucho)
- Sistemas de procesamiento de streaming (ej.: Storm)
- Sistemas de encuestas anónimas por Internet
- Timeline de una red social
- Sistema de mensajería para grupos

Otros conceptos

- Strong consistency
- Weakly consistent
- Eventual consistency
- Causual consistency

- También se aplican a estructuras de datos

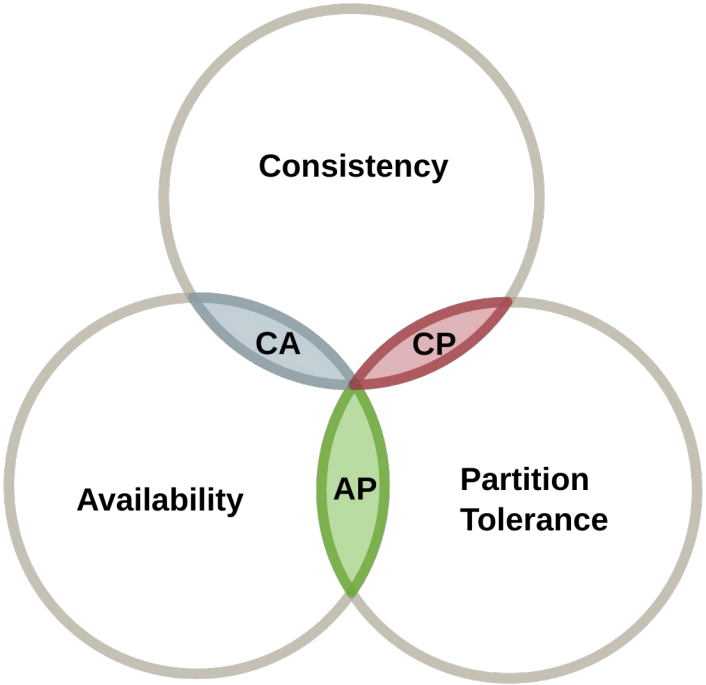
Teorema CAP

- En *teoría de Sistemas Distribuidos*, el teorema CAP dice que es imposible conseguir (al 100%) las siguientes tres propiedades (en un SD):
  - **C**onsistencia → Todos los nodos ven los mismos datos, al mismo tiempo
  - Disponibilidad (**A**vailability) → Cada pedido recibe una respuesta sobre si algo se ha llevado a cabo de manera exitosa o ha fallado
  - Tolerancia a **p**articiones → El sistema continúa funcionando a pesar de que ha ocurrido una partición arbitraria a causa de fallos en la red
- CAP ⇒ Consistencia afecta latencia
- CAP ⇒ Es muy difícil conseguir las tres propiedades simultáneamente
  - 2 de 3 es más fácil
- Decisiones CAP nos dicen qué hace el sistema cuando hay una partición

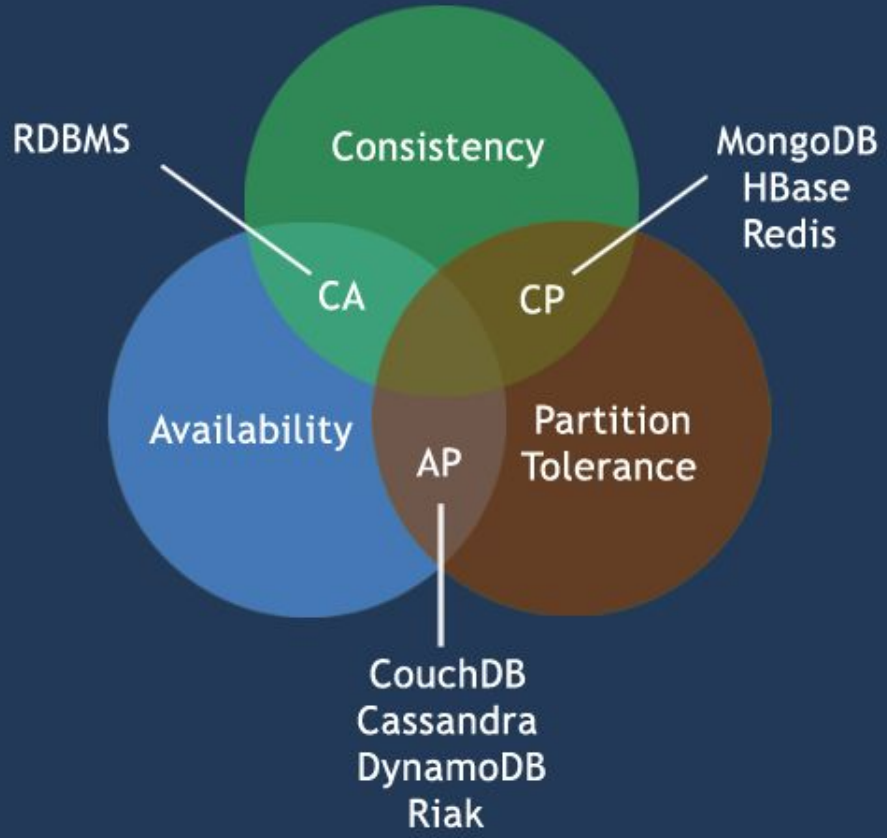
Ejemplo: Sometimes it makes sense to forfeit strong C to avoid the high latency of maintaining consistency over a wide area. Yahoo's PNUTS system incurs inconsistency by maintaining remote copies asynchronously.5 However, it makes the master copy local, which decreases latency. This strategy works well in practice because single user data is naturally partitioned according to the user's (normal) location. Ideally, each user's data master is nearby.

Ejercicio

Liste 1-2 SD para cada una de las intersecciones CAP:

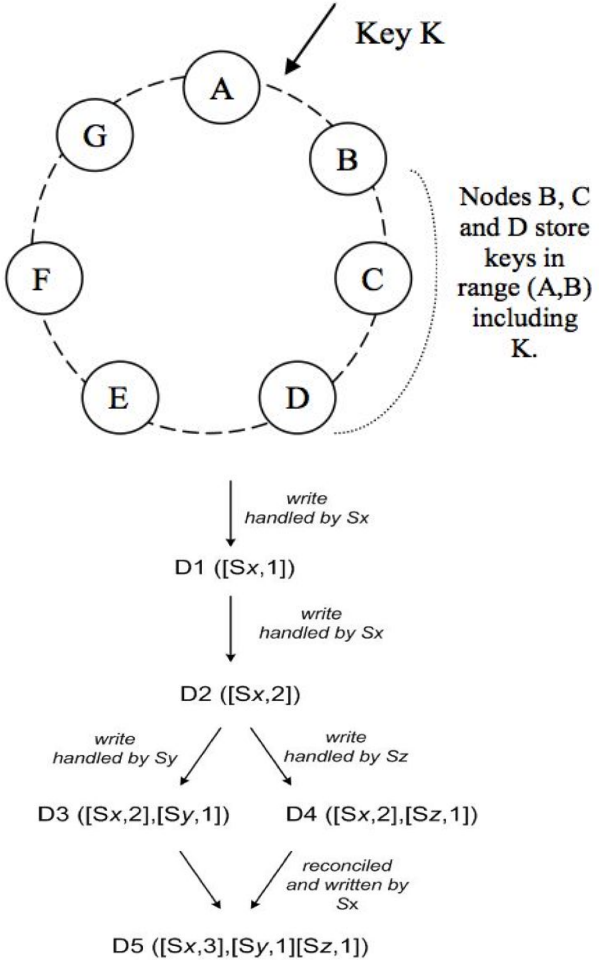


CAP Theorem


RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

amazon DynamoDB Architecture

- True distributed architecture
- Data is spread across hundreds of servers called storage nodes
- Hundreds of servers form a cluster in the form of a "ring"
- Client application can connect using one of the two approaches
  - Routing using a load balancer
  - Client-library that reflects Dynamo's partitioning scheme and can determine the storage host to connect
- Advantage of load balancer – no need for dynamo specific code in client application
- Advantage of client-library – saves 1 network hop to load balancer
- Synchronous replication is not achievable for high availability and scalability requirement at amazon
- DynamoDB is designed to be "always writable" storage solution
- Allows multiple versions of data on multiple storage nodes
- Conflict resolution happens while reads and NOT during writes
  - Syntactic conflict resolution
  - Symantec conflict resolution



Version evolution of an object over time

ACM SOSP 2007

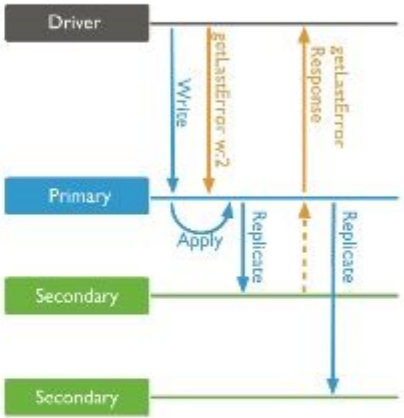
Cluster: MongoDB

- CAP theorem
  - >If the network is down choose
  - >Consistency xor
  - >Availabilty
- Deals with replication
- MongoDB has master / slave replication

- Write Concerns:
  - >Unacknowledged
  - >Acknowledged
  - >Journaled
  - >Some nodes in the replica set
- Queries might go to master only or also slaves
- Influences consistency

Write concern **replicas_safe** or **replica_acknowledged**

- Waits for at least 2 servers for the write operation



The findings

- 6000 total writes
- 5695 acknowledged
- 3768 survivors
- 1927 acknowledged writes lost (33% write loss)

Mongo only verifies that a write took place against two nodes.

A new primary can be elected without having seen those writes.

In this case, Mongo will rollback those writes.

Lectura recomendada:
https://aphyr.com/posts/284-call-me-maybe-mongodb

Decisión de Diseño:
Servicios con estado o sin estado

Stafeful vs Stateless

Con estado vs. sin estado

- Estado → cualquier información con respecto a las interacciones en curso
  - ¿Ejemplos de estado?
- Con estado (stateful)
  - Mantiene estado de clientes conectados
  - Ej.: Sesiones en servidores web
- Sin estado (stateless)
  - No mantiene estado para cada cliente
- Alternativa: soft state
  - Mantiene estado por un tiempo limitado; descartar estado no tiene impacto en “correctness”
- El **protocolo** es quien determina si implementación es con o sin estado

Ejemplo: Servidor de archivos con estado

Diseño:

- **Server provides *handle* to access state at the server**
- **File open**
  - **Request: OPEN(/tmp/test.txt, “r”)**
  - **Reply: OPEN(ok, 32) -- handle = 32**
  - **State: 32: /tmp/test.txt, 0, read**
- **File read**
  - **Request: READ(32, 200)**
  - **Reply: READ(ok, data)**
  - **State: 32: /tmp/test.txt, 200, read**

¿Se lo puede convertir a un servidor sin estado?

Ejercicio

Analice y liste las ventajas y desventajas de una implementación de un servicio *con estado* (vs. una sin estado).

Aspectos de diseño para protocolos con estado

- Time-outs
- Pedidos y respuestas duplicados
  - Operaciones deben ser idempotentes
  - O servidor debe poder manejar duplicados sin conllevar a inconsistencias
- Fallas crash (de cualquiera de los dos extremos)
- Manejo de múltiples clientes
  - ¿Dónde se mantiene el estado de cada cliente?
    - ¿Problemas?
- Bloqueo de recursos (sincronización)

Diseños reactivos (iniciados por eventos)

Diseños reactivos

- Es necesario contruir sistemas distribuidos que:
  - Reaccionen a eventos → Event-driven
  - Reaccionen a la carga → Escalables y elásticos
  - Reaccionen a fallas → Resilientes
  - Reaccionen a usuarios → Responsive
- ⇒ Aplicaciones reactivas
- Definiciones
  - *Reactivo.- Que responde rápidamente a un estímulo*
  - *Event-driven.- El flujo de un programa está determinado por eventos*
  - *Escalable.- Con capacidad de expandirse o mejorarse fácilmente y bajo demanda*
  - *Elástico.- Auto-Escalable (y auto-reducible)*

Las 8 falacias de los sistemas distribuidos

1. La red es confiable
2. La latencia es cero
3. Ancho de banda es infinito
4. La red es segura
5. La topología no cambia
6. Hay un solo administrador
7. Costo de transporte es cero
8. La red es homogénea

⇒ Usar paso de mensajes asíncrono

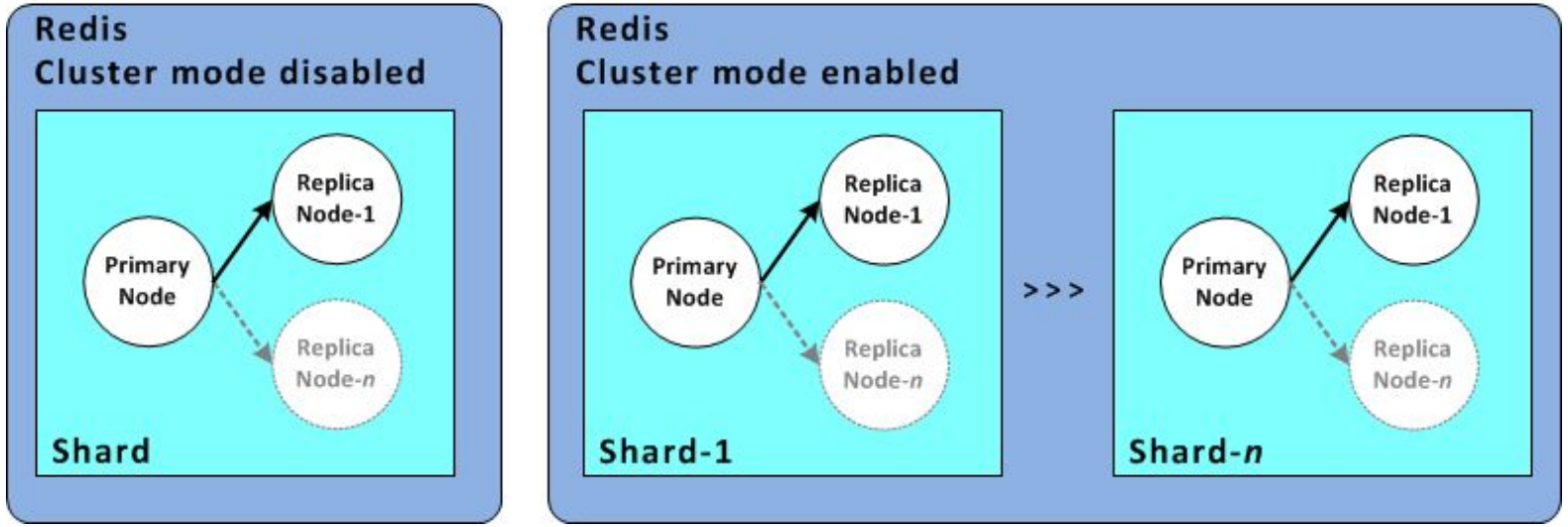
Recomendaciones

- Evitar tener estado compartido en implementaciones multi-hilos
  - “Synchronization is hard”
  - Resultado ⇒ Condiciones de carrera ⇒ Ejecuciones no determinísticas
- ¡Nunca bloquearse! (a menos que sea estrictamente necesario)
  - Reduce escalabilidad y rendimiento
  - No acaparar recursos que no se están usando
  - Usar invocaciones sin bloqueo
  - Usar concurrencia libre de candados/semáforos
- Particionar y replicar
  - Particionar otorga escalabilidad
  - Replicación otorga tolerancia a fallos

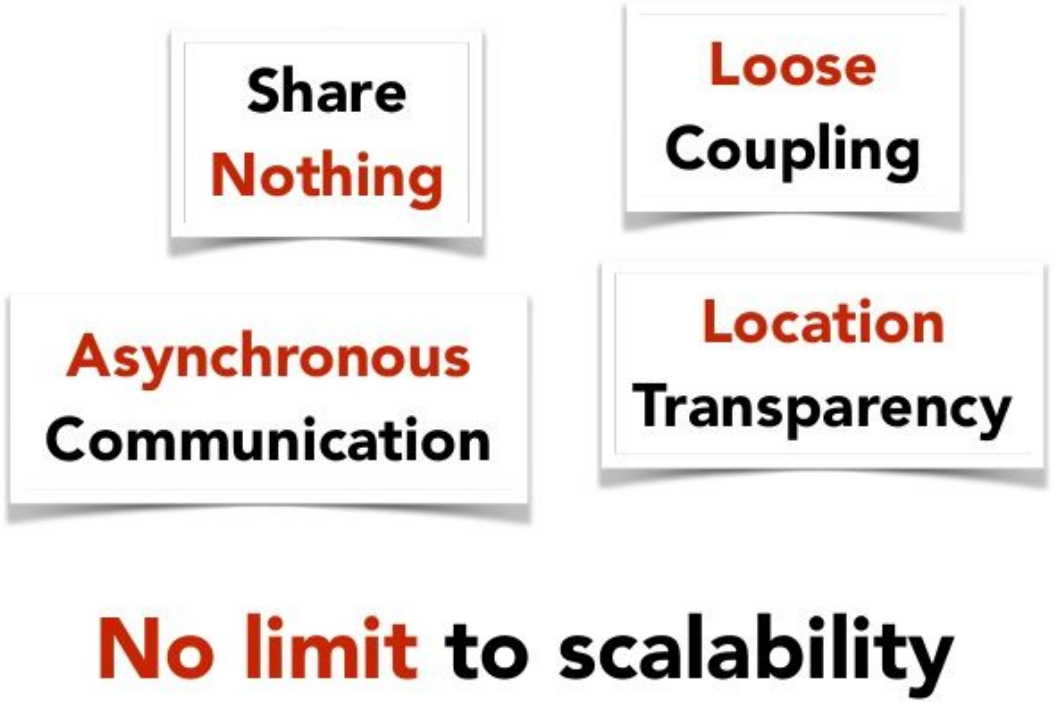
Ejemplo: Particionamiento y replicación en AWS ElastiCache

Redis (cluster mode disabled)
Supported by Redis 2.8.x and 3.2.x
Replication
Single shard
Modifiable
No data partitioning

Redis (cluster mode enabled)
Supported by Redis 3.2.x
Replication within each shard
Multiple shards
Static/not modifiable
Data partitioning supported

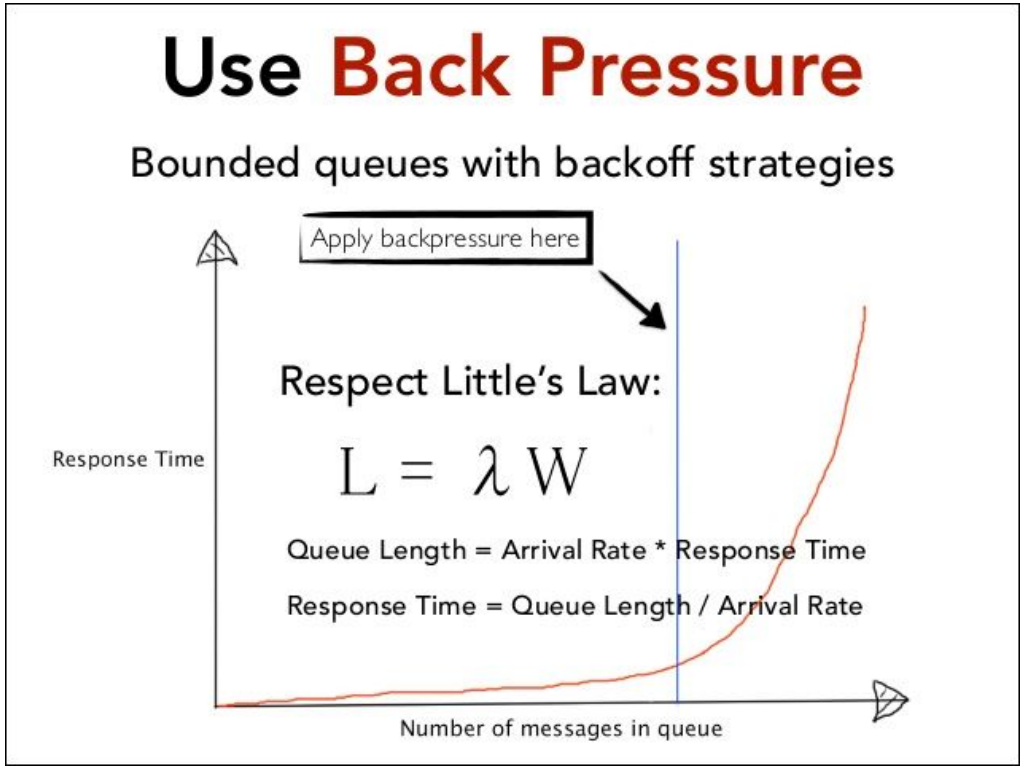


Recomendaciones (cont.)



Desafío: Latencia consistente

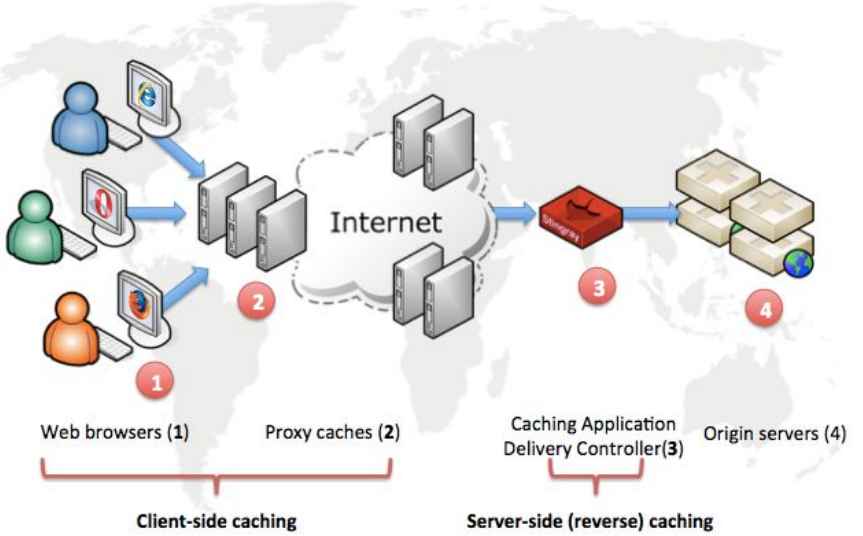
- Escenarios ideales
- Picos (bursts) de tráfico
- Fallas



Localidad de Datos

Importancia de la localidad de datos

- Transferencia de datos afecta rendimiento de sistemas distribuidos
  - Latencia y throughput
- Lo recomendable → Usar datos locales
- ¿Cómo?
  - Mover cómputo a los datos
  - Replicar datos
  - Cachear datos



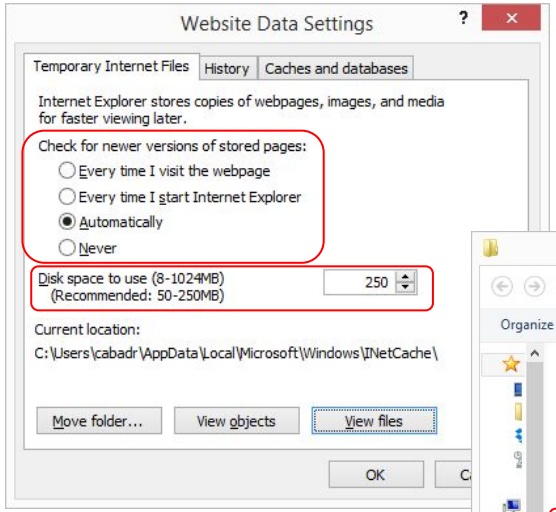
Cacheo

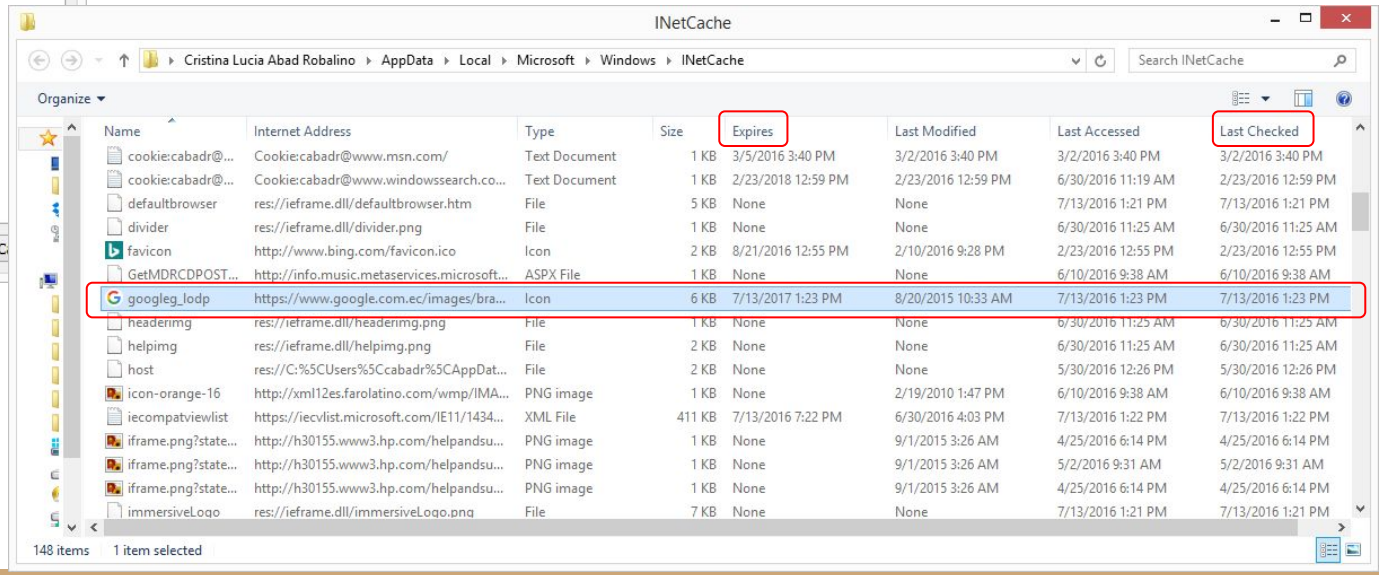
- Caché → sitio de escondite usado especialmente para almacenar provisiones
- Cachear datos:
  - Esconde la latencia
  - Mejora el rendimiento en accesos repetidos
- Problema:
  - Problemas de consistencia de la caché

Ej. manejo de consistencia en el disk cache

- Write-through
  - ¿Y si un cliente lee datos desactualizados?
    - Alternativa 1: Contactar al servidor para cada acceso
    - Alternativa 2: Servidor envía invalidaciones a las cachés
- Write-behind
  - Cachear escrituras localmente en un buffer → Escrituras en “bulk” (paquete)
  - Problema: Semánticas ambiguas
- Read-ahead (prefetch)
  - Pedir datos por adelantado → Minimiza latencia
- Write-on-close
  - Admitir que tenemos semánticas de sesión
- Control centralizado
  - Coordinador lleva registros de modificaciones y controla escrituras (analogía: señalización)

Ejemplo: Cacheo local

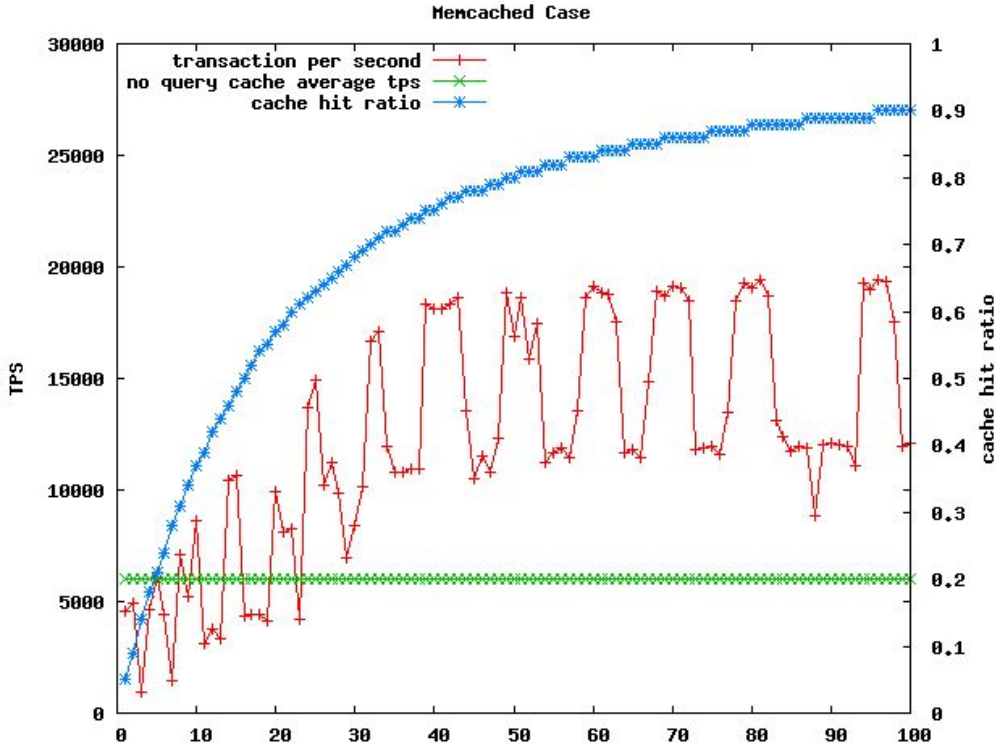




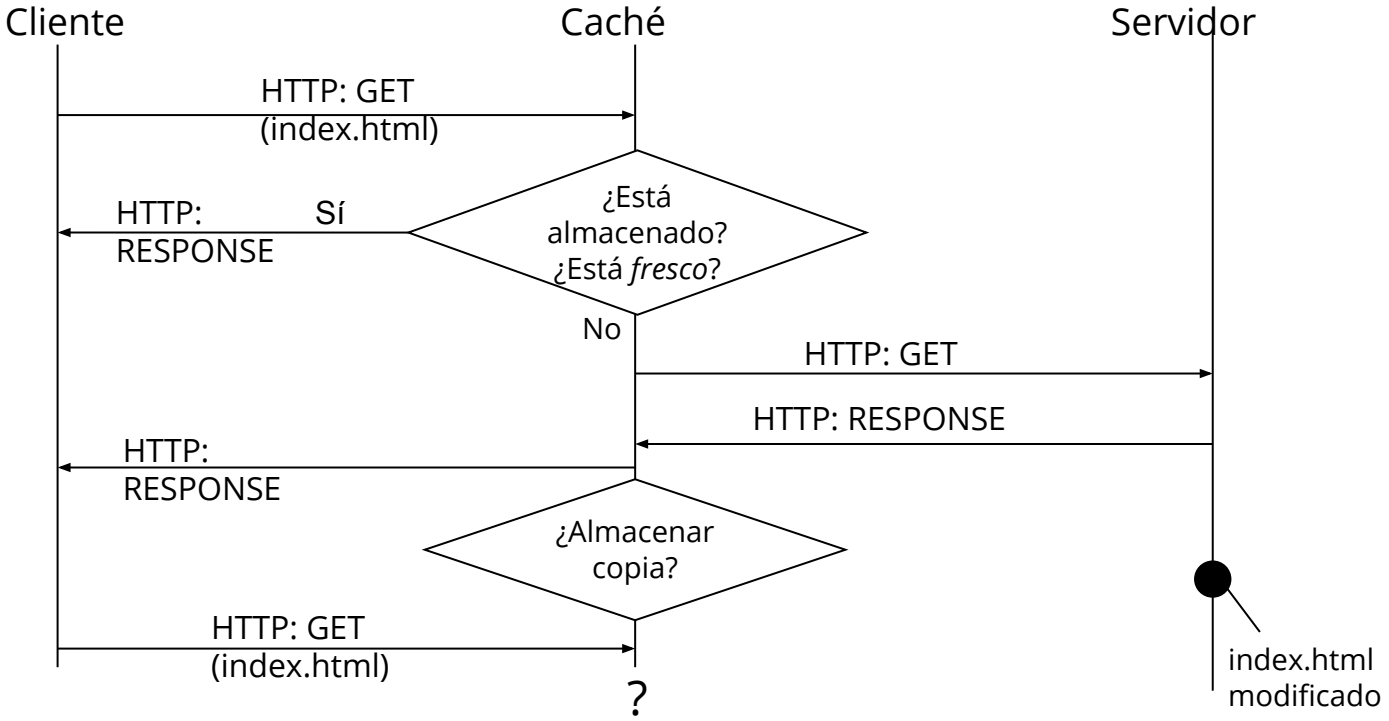
Parámetros que afectan rendimiento de una caché

- Tamaño máximo
- Reglas de cacheabilidad
- Latencia de acceso a la caché
- Throughput de la caché (tasa de transacciones máximas)
- Ubicación
- Características del contenido pedido por los usuarios
    - Cargas de trabajo (workloads)

Ejemplo: Postgres + Memcached



Ejemplo: Caché web compartida



Políticas de reemplazo (Cache eviction policies)

- También llamadas: Políticas de desalojo
- Esquema general usado para decidir qué objetos se eliminan de la caché cuando está llena y nuevos objetos necesitan almacenarse
- Reglas Dinámicas de Objetos: usadas para examinar las características de un objeto y estimar su valor futuro para determinar si vale la pena añadirlo a la caché
- ¿Ejemplos?

Políticas de reemplazo (Cache eviction policies)

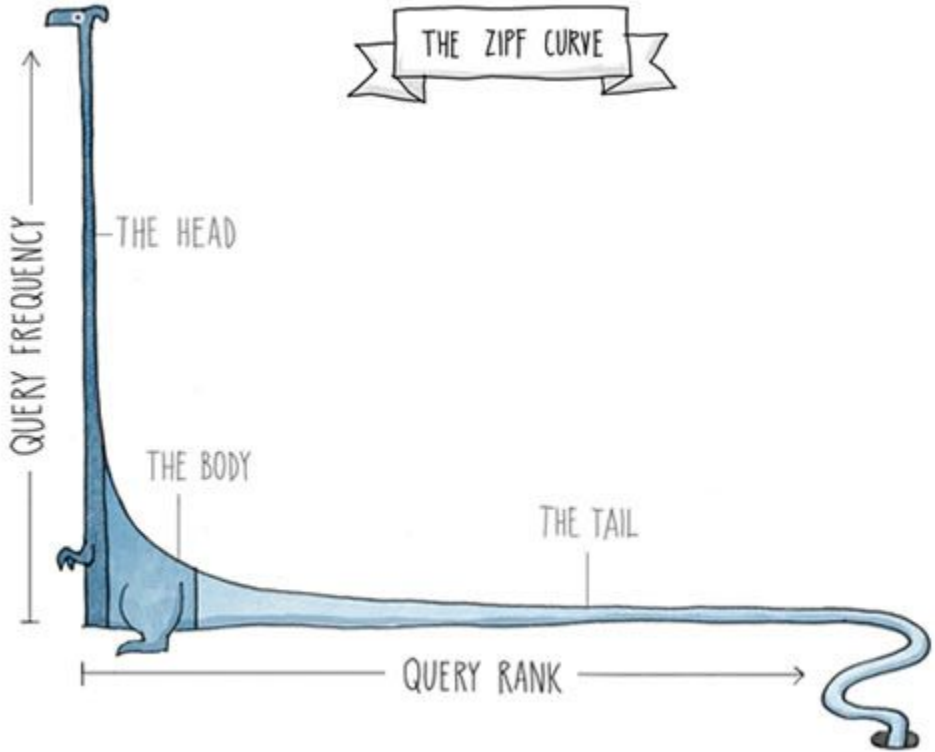
- LRU (least recently used)
- FIFO (first in first out)
- LFU (least frequently used)
- NTE (next to expire)
- LFF (largest file first)
- RANDOM

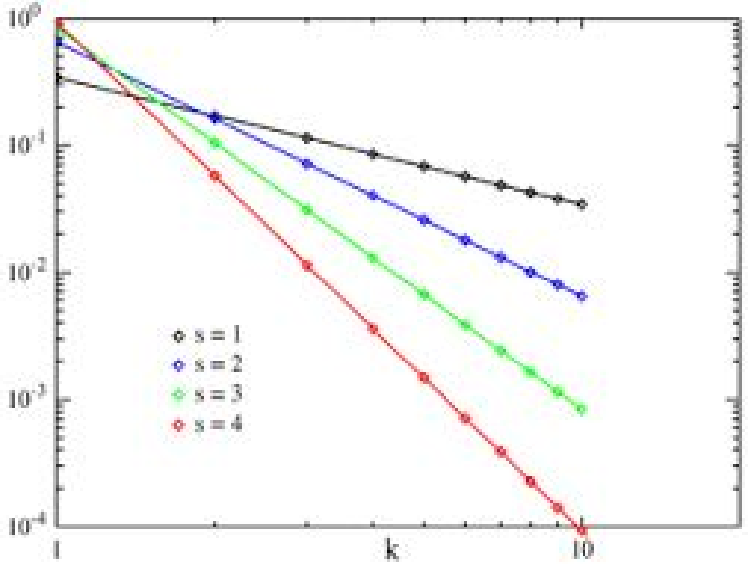
Generalmente se combinan y complementan con heurísticas para proporcionar soluciones prácticas

Sin embargo, es área de investigación activa ⇒ Mejoras continuas

Ej.: Uso de Bloom filters

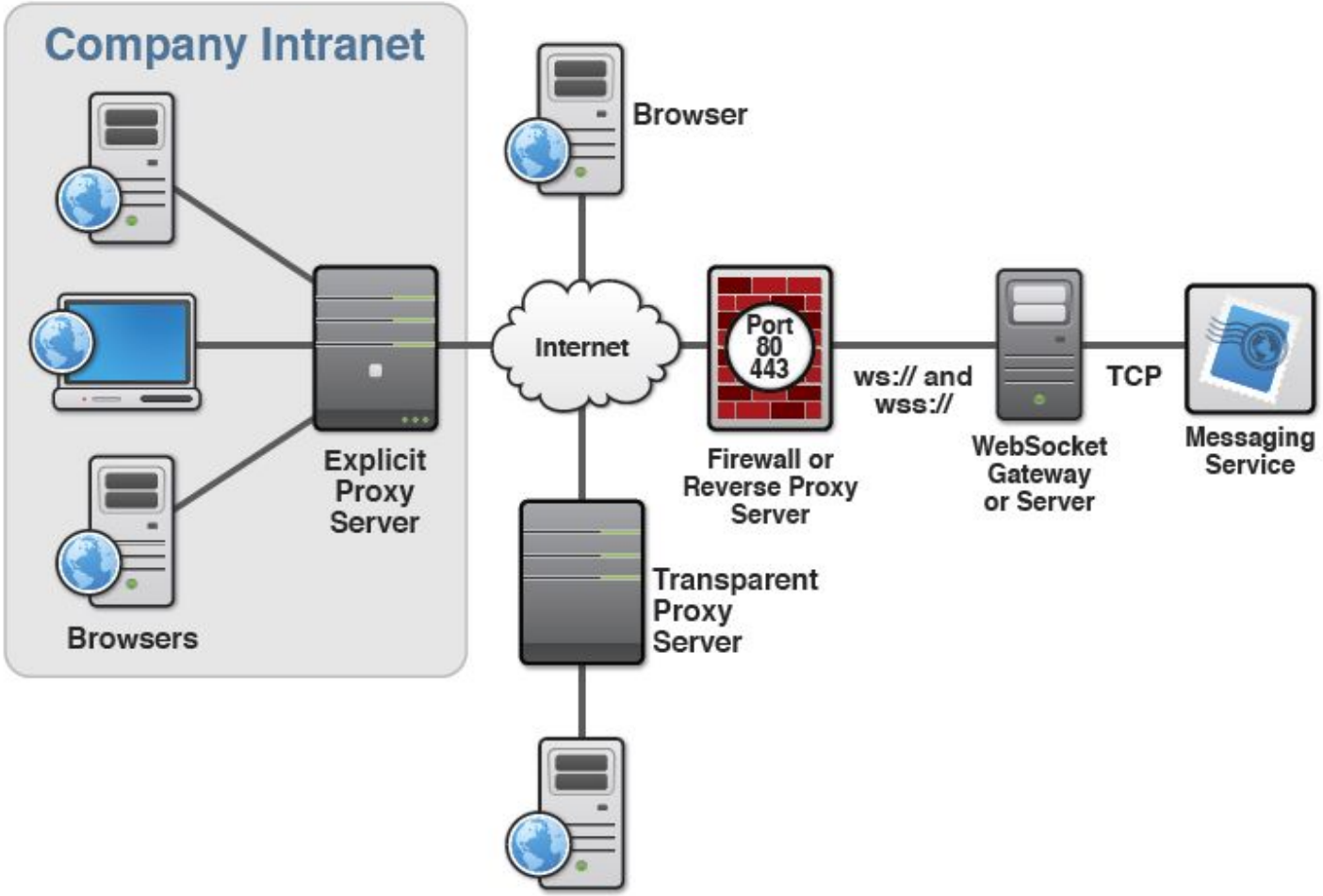
Zipf


THE ZIPF CURVE
QUERY FREQUENCY
THE HEAD
THE BODY
THE TAIL
QUERY RANK


10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
s = 1
s = 2
s = 3
s = 4
1
k
10

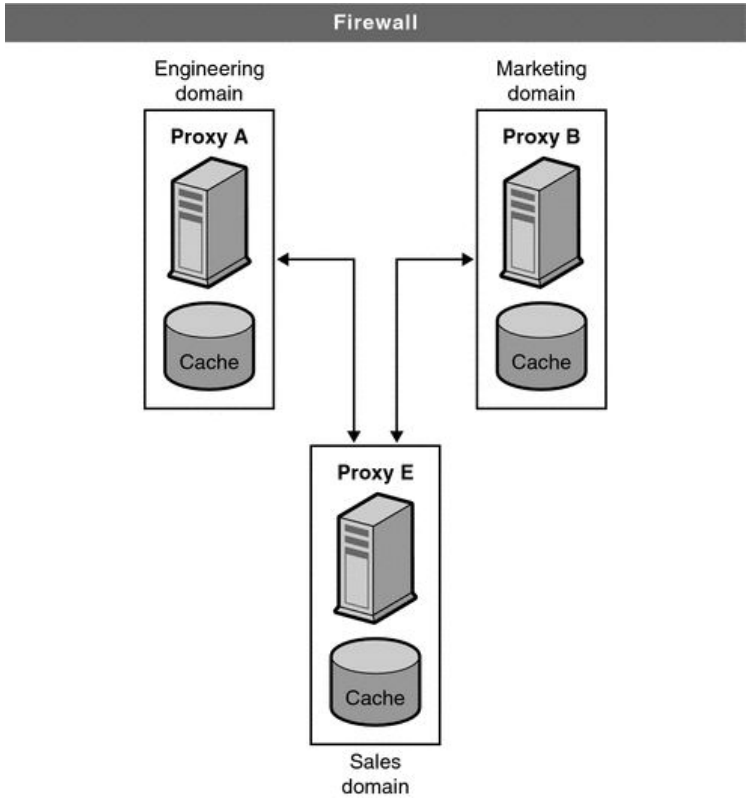
Ubicación de una caché en la red

- Forward proxy
  - Actúa en representación de un grupo de consumidores de contenido
  - En cliente se configura en Internet Options → Connections → LAN Settings → Proxy
- Proxy de intercepción o proxy transparente
  - Sirve el tráfico de red que se le direcciona
  - Ej.: caché de un ISP (transparente para el cliente)
- Reverse proxy o acelerador de servidor
  - Actúa en representación del servidor de origen y ayuda a un grupo específico de servidores
  - Ej.: Google, Facebook (transparente para el cliente)


Company Intranet
Browsers
Explicit
Proxy
Server
Internet
Browser
Transparent
Proxy
Server
Port
80
443
Firewall or
Reverse Proxy
Server
ws:// and
wss://
WebSocket
Gateway
or Server
TCP
Messaging
Service

Cachés jerárquicas

- Se comunican entre sí usando Internet Caching Protocol (ICP)
- Ej.: Squid, Varnish



Caso de estudio: Redes de distribución de contenidos

Facultando las CDNs

- DNS para compartir cargas
- Switching en capas 4-7
- Ruteo global de pedidos

DNS para compartir cargas: DNS round-robin

- Configurar varias IP para cada nombre
  - Name server rota (round-robin) entre estos al contestar consultas DNS
  - TTL: tiempo de vida de los registros cacheados en clientes
- Desventajas
  - TTL limita la rotación
  - Sirve para **compartir** cargas pero no las **balancea**
  - Cacheo en proxies y clientes → resultados no predecibles
  - No sabe el contexto bajo el cual se hizo el pedido; esta info. podría servir para dar un mejor servicio

Ejemplo

```
[cabad@server1 ~]$ nslookup www.stackoverflow.com
Server:         192.188.59.2
Address:        192.188.59.2#53

Non-authoritative answer:
www.stackoverflow.com   canonical name = stackoverflow.com.
Name:   stackoverflow.com
Address: 151.101.65.69
Name:   stackoverflow.com
Address: 151.101.129.69
Name:   stackoverflow.com
Address: 151.101.193.69
Name:   stackoverflow.com
Address: 151.101.1.69

[cabad@server1 ~]$
```

Ejemplo (cont.)

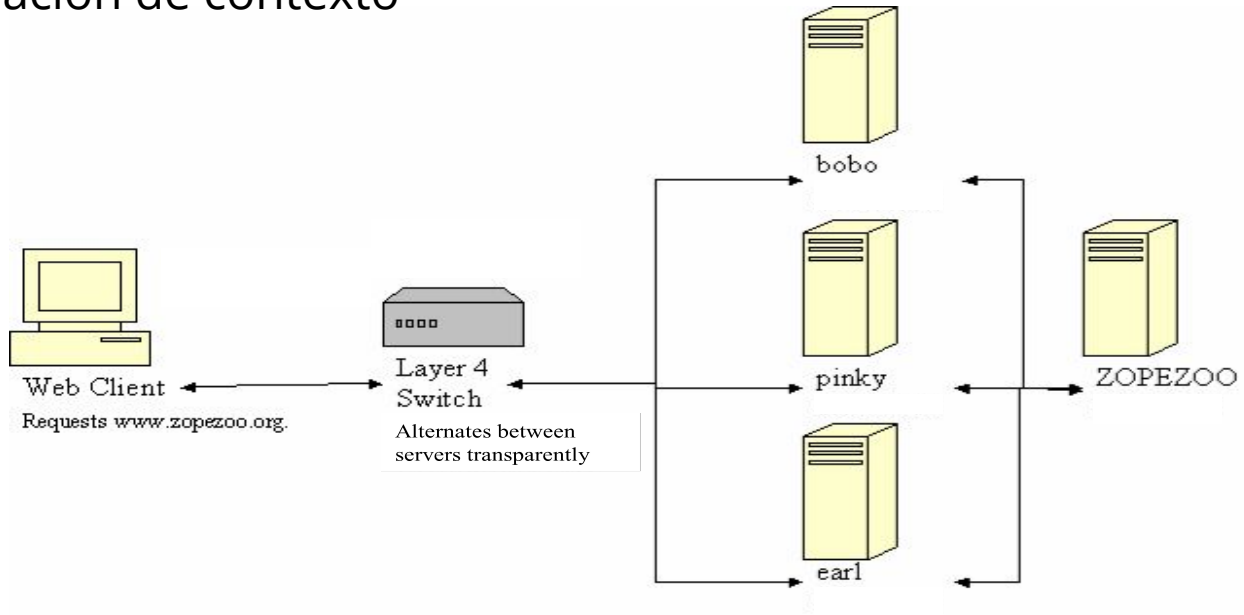
```
[cabad@server1 ~]$ nslookup www.stackoverflow.com
Server:         192.188.59.2
Address:        192.188.59.2#53

Non-authoritative answer:
www.stackoverflow.com   canonical name = stackoverflow.com.
Name:   stackoverflow.com
Address: 151.101.129.69
Name:   stackoverflow.com
Address: 151.101.193.69
Name:   stackoverflow.com
Address: 151.101.1.69
Name:   stackoverflow.com
Address: 151.101.65.69

[cabad@server1 ~]$ ping www.stackoverflow.com
PING stackoverflow.com (151.101.65.69) 56(84) bytes of data.
64 bytes from 151.101.65.69: icmp_seq=1 ttl=54 time=61.3 ms
^C
--- stackoverflow.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 338ms
rtt min/avg/max/mdev = 61.394/61.394/61.394/0.000 ms
[cabad@server1 ~]$ ping www.stackoverflow.com
PING stackoverflow.com (151.101.1.69) 56(84) bytes of data.
64 bytes from 151.101.1.69: icmp_seq=1 ttl=54 time=61.9 ms
^C
--- stackoverflow.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 247ms
rtt min/avg/max/mdev = 61.900/61.900/61.900/0.000 ms
[cabad@server1 ~]$ ping www.stackoverflow.com
PING stackoverflow.com (151.101.193.69) 56(84) bytes of data.
64 bytes from 151.101.193.69: icmp_seq=1 ttl=54 time=62.4 ms
```

Switching en capas 4-7

- Como DNS round robin, pero permiten balancear carga y usar información de contexto



- Permite que servidores tengan IP privada

Switching en capas 4: Balanceo de carga

- Variedad de políticas de balanceo
  - Mejor disponible
    - Para pedidos nuevos
  - Persistencia
    - Conectar al mismo cliente con el mismo servidor
  - Servicios diferenciados
    - Diferentes clases de usuarios pueden recibir diferente servicio

Políticas de Mejor Disponible ⟶ ¿Cuál es el “mejor”?

- Elección aleatoria del servidor → Distribución uniforme
- Round Robin → Distribución uniforme
- Distribución ponderada (estática)
  - Administrador puede especificar % de tráfico que debe ir a cada servidor
- Distribución ponderada (dinámica)
  - Se dirige más tráfico a los servidores con mejor tiempo de respuesta
- Menos conexiones
- Menos paquetes (recibidos recientemente)
- Servidor menos ocupado
  - Agente en servidor monitorea su capacidad e informa al switch

Switching en Capa 7

- Switches en capa 7 usan información como URL, cabecera HTTP, cookies, identificadores de sesión SSL
- Uso:
  - Dedicar servidores para usos específicos: ej.: servidor de imágenes, o servidor de páginas dinámicas (servidores pueden estar *tuned* apropiadamente)
  - Usar información de cacheabilidad (cabecera HTTP) para determinar si re-enviar el pedido al proxy de intercepción o no
  - Aplicación de políticas en base a cookies

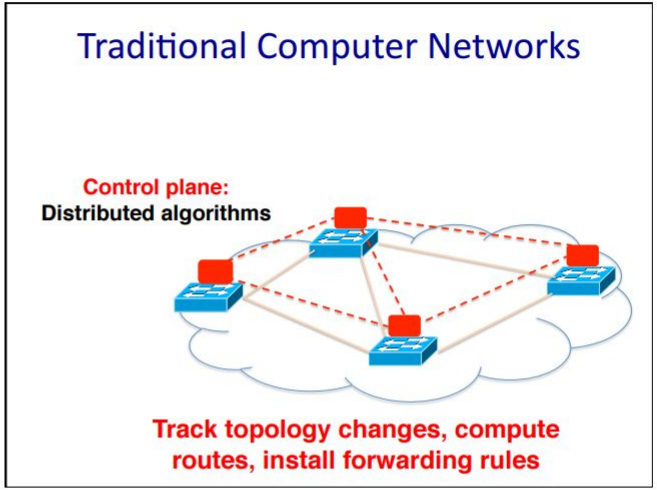
Terminología: Middleboxes

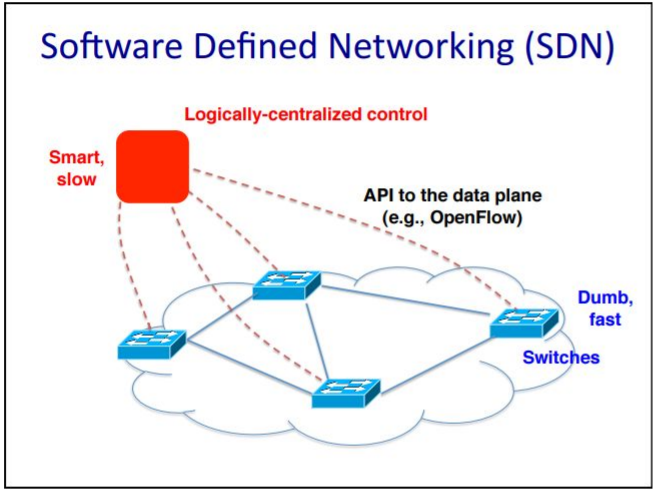
- Las **middleboxes** son intermediarias
  - Interpuestas entre dos hosts que se están comunicando
  - Generalmente sin que una o ninguna de las partes lo sepan
- Ejemplos:
  - Network Address Translators
  - Firewalls
  - Traffic shapers
  - Intrusion detection systems
  - Transparent Web proxy caches
  - Application accelerators

Referencia: http://www.cs.princeton.edu/courses/archive/spr09/cos461/docs/lec08-middlebox.pdf

Terminología: SDN (Software Defined Networks)

- OpenFlow
  - Permite crear reglas simples para manejar paquetes
  - Unifica equipos esp.: Router, switch, firewall, NAT
- Ejemplos de aplicaciones:
  - Dynamic access control
  - Seamless mobility/migration
  - Server load balancing
  - Network virtualization
  - Using multiple wireless access points
  - Energy efficient networking
  - Adaptive traffic monitoring
  - Denial of Service attack detection





Referencia: https://www.cs.princeton.edu/courses/archive/spring12/cos461/docs/lec24-sdn.pdf

Análisis

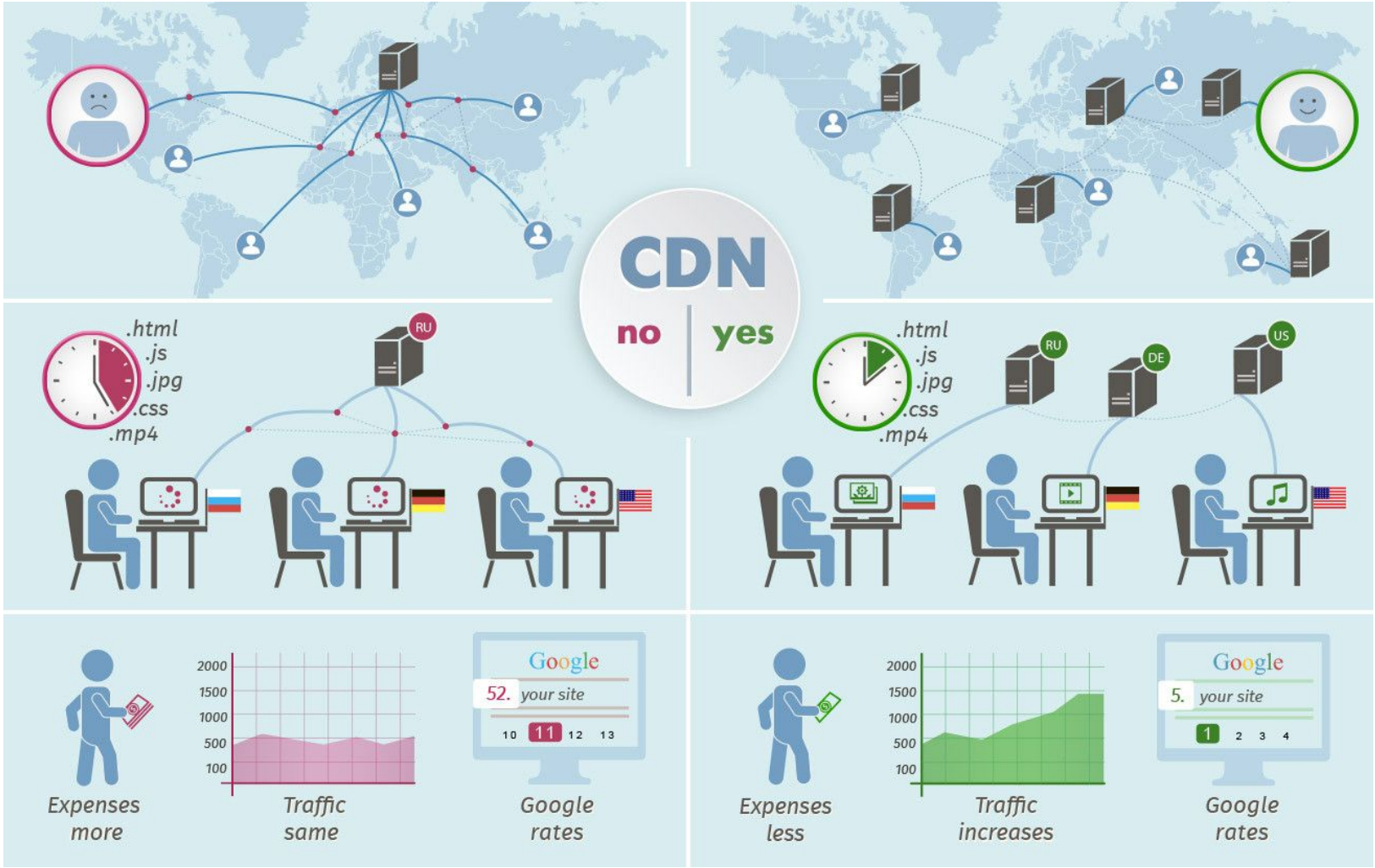
¿Qué switch es (potencialmente) más rápido? ¿Uno de capa 4 o uno de capa 7? ¿Por qué?

Proxies de intercepción

- Un switch Web es clave para el funcionamiento correcto de una configuración con proxies de intercepción
  - Para que no todo pedido vaya al proxy
    - Puede usar reglas de capa 4 (ej.: puerto 80)
    - O, de capa 7 (ej.: información cacheable)
  - Para mejor rendimiento se puede tener cluster de proxies y el Web switch balancea la carga entre ellos

Ruteo global de pedidos

- Para mejorar rendimiento yo puedo querer re-direccionar pedidos a servidores (réplicas) en otros lugares geográficos
- Desafíos
  - ¿Cómo re-direccionar a clientes?
  - ¿Cómo estimar cercanía?
- CDN: Content Delivery/Distribution Networks
  - Redes de distribución de contenidos


CDN
no
yes
.html
.js
.jpg
.css
.mp4
RU
DE
US
2000
1500
1000
500
100
Google
52. your site
10 11 12 13
5. your site
1 2 3 4
Expenses more
Traffic same
Google rates
Expenses less
Traffic increases
Google rates

Re-direccionando pedidos de clientes

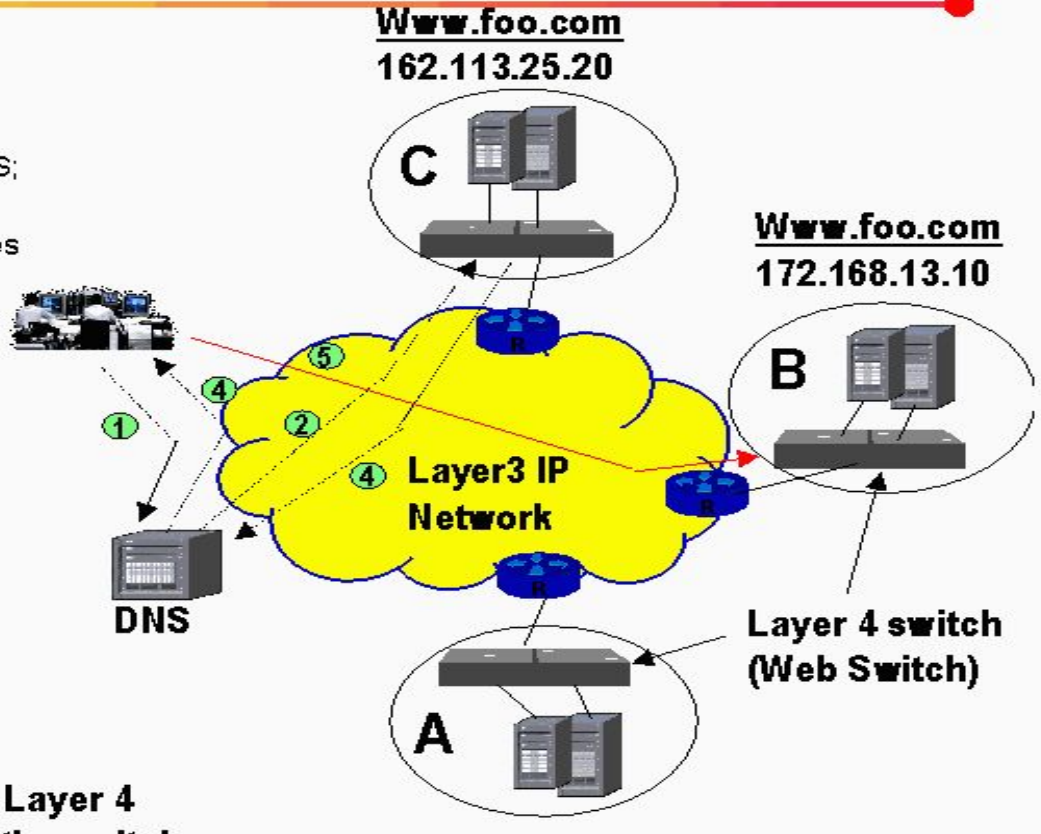
- Balanceo global de cargas de servidores (GSLB)
- Ruteo de pedidos basado en DNS
- Re-escribiendo HTML

Balanceo Global de Cargas de Servidores

- Proporcionado por algunos fabricantes de switches Web
- Extensión a switches de capas 4-7 en el cual los switches se comunican con otros distribuidos globalmente para estar conscientes de la situación de los servidores locales y los remotos; además, combinan esto con comportamiento de DNS
  - Por ejemplo algunos switches también combinan info. de cercanía; el switch de manera pasiva recopila info. de las redes, para saber cuál está más cerca de cada granja de servidores

Layer 4 Switching for Global Load Balance and Traffic Redirect *(An Example - DNS Redirect)*

1. Client DNS request for www.foo.com sent to local DNS;
2. Local DNS quires upstream DNS at Site C;
   Web switch at Site C gets DNS request and determines that Site B is best to serve this user's request;
3. Acting as Authoritative Name Server, web switch returns Site B's IP address to client's local DNS;
4. Local DNS server responds to client with IP address for Site B's server farm;
5. Client opens application session to 172.168.13.10 (Site B);
   Local load balancing at Site B selects best local server.



Note: Some of current commercially available Layer 4 switches integrate DNS functionality in the switch

¿Cómo se selecciona la mejor ubicación geográfica?

- En base a chequeos periódicos de *salud*
- En base a chequeos *flashback* en tiempo real (pings)
- En base a info. geográfica (ya que IANA asignó ciertas IP por continente)
- Condiciones de carga del servidor
- Heurísticas configuradas por el administrador
- Estimaciones de proximidad de los switches Web a los clientes (pasivo, en base a RTT)

Re-escribiendo HTML

- No se re-direcciona el primer pedido sino pedidos subsiguientes
- Re-escritura puede ser
  - *a priori*: cuando yo hago pedido el servidor decide cuál versión del HTML me envía
  - *Bajo demanda:* HTML es re-escrito para cada cliente/pedido
- Ej.: Akamai


```
</script><script type="text/javascript">
                                var mobileDomain = 'm.www.1800flowers.com';
                                var subDomain    = 'www';
                                var mobileStartDomain = 'm';
                                setMobileURL();
                        </script><script type="text/javascript">
                        var seoUrlForHomeColProdPage = '';
                        setUpSEOUrlForHomeColProdPage(seoUrlForHomeColProdPage);
                </script><meta property="fb:app_id" content="222074941478100">
<meta property="og:type" content="product_service">
<meta name="google-site-verification" content="IcSMkCX7So6-JyimbXKDp43SRV05i1R_oiyFKK5PKQk">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Flowers | Flower Delivery | Fresh Flowers Online | 1-800-FLOWERS.COM</title>
<meta name="description" content="Send flowers and send a smile! Discover fresh flowers online, gift baskets, and flo
delivery is easy at 1-800-FLOWERS.COM.">
<meta name="keywords" content="flowers, send flowers, flower delivery, fresh flowers, flowers online, flower arrangem
flower, florist, florists">
<link rel="canonical" href="https://www.1800flowers.com">
<link href="https://m.www.1800flowers.com" media="only screen and (max-width:640px)" rel="alternate">
<link rel="stylesheet" href="//cdn2.1800flowers.com/wcsstore/Flowers/ww42/css/flowers-mbp.css" type="text/css">
<link rel="stylesheet" href="//cdn2.1800flowers.com/wcsstore/Flowers/ww42/css/flowers-ext-mbp.css" type="text/css">
<!--[if IE]><link rel="stylesheet" href="//cdn1.1800flowers.com/wcsstore/Flowers/ww42/css/flowers-ie.css" type="text/
type="text/javascript">
        djConfig = { isDebug: false, parseWidgets:false, searchIds:[]};
        </script><script>
var logonFormURLJS = '/LogonForm?catalogId=13302&amp;langId=-1&amp;storeId=20051&amp;remember=true&amp;URL=';
</script><style id="antiClickjack">body{display:none;}</style>
<script type="text/javascript">
if (self === top) {
   var antiClickjack = document.getElementById("antiClickjack");
   if(null != antiClickjack){
```

Estimando la Proximidad

- Sondeo reactivo
  - Introduce latencia adicional
- Sondeo proactivo
  - Monitoreo puede molestar a DNSs
- Monitoreo de conexiones
  - Totalmente pasivo (SYN/ACK – ACK)

Ejercicios

1. Explique por qué razón el hit rate (tasa de pedidos que pueden ser satisfechos directamente de la caché) es probablemente más alto en un reverse proxy (instalado en el sitio de un servidor Web) que en un proxy de intercepción instalado en un ISP.
2. DNSs y Akamai (ver siguiente diapositiva)

a. Google and OpenDNS offer alternative DNS servers that they claim are more reliable, better provisioned, and with better cache hit ratios than your ISP's DNS server. However, using these resolvers can result in sub-optimal interaction with CDNs like Akamai.

As an example, using my regular resolver at Brown CS, I get the following answer to looking up www.bestbuy.com:

```
www.bestbuy.com.    3552    IN  CNAME   www.bestbuy.com.edgesuite.net.
www.bestbuy.com.edgesuite.net. 5610 IN  CNAME   a1105.b.akamai.net.
a1105.b.akamai.net. 3   IN  A   198.7.236.240
```

Pinging 198.7.236.240, I get an average latency of 0.3ms, and traceroute reveals that this server is located in Oshean's network, which is Brown's network provider.

If I use Google's resolver instead, 8.8.8.8, I get the following answer:

```
www.bestbuy.com.    2169    IN  CNAME   www.bestbuy.com.edgesuite.net.
www.bestbuy.com.edgesuite.net. 19535 IN CNAME   a1105.b.akamai.net.
a1105.b.akamai.net. 11  IN  A   208.44.23.16
```

Pinging 208.44.23.16, I get an average of 26ms, and a path that has to cross at least two more ASes. **Why does Akamai give me a worse server when I use a different resolver?** [5 pts]

Lectura recomendada

Nygren, Erik, Ramesh K. Sitaraman, and Jennifer Sun. **"The Akamai network: A platform for high-performance internet applications."** *ACM SIGOPS Operating Systems Review* 44.3 (2010): 2-19.

https://www.akamai.com/it/it/multimedia/documents/technical-publication/the-akamai-network-a-platform-for-high-performance-internet-applications-technical-publication.pdf

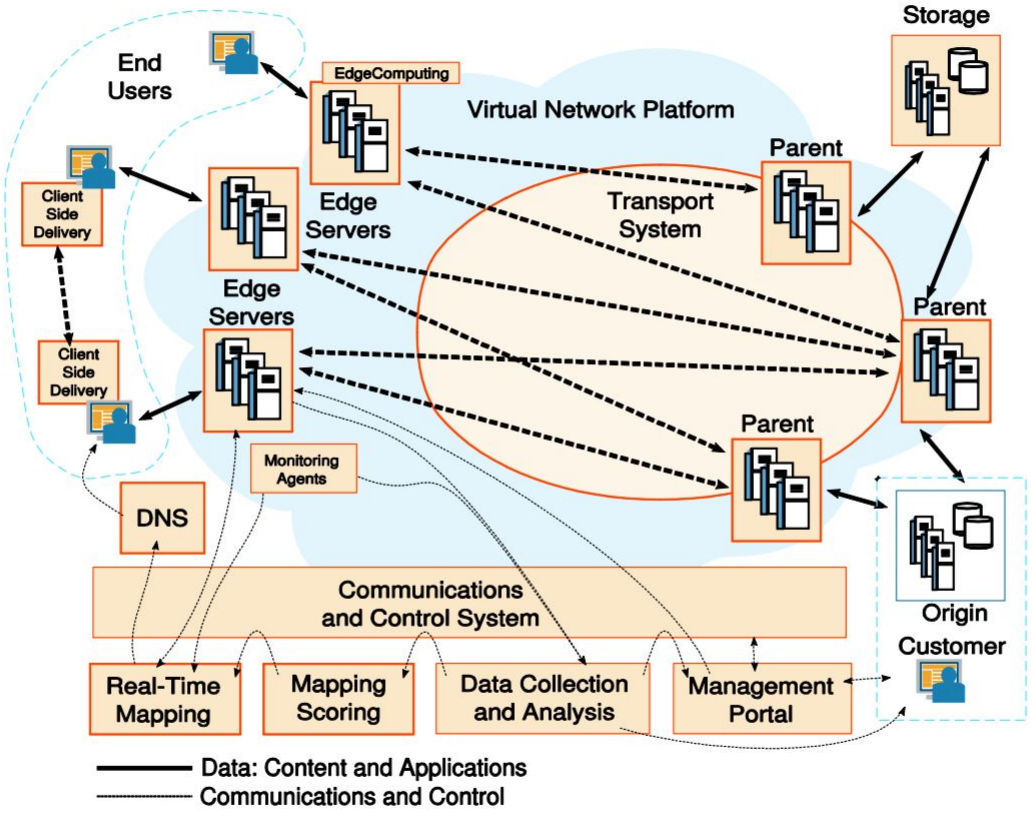


Figure 5: System components of the Akamai platform.