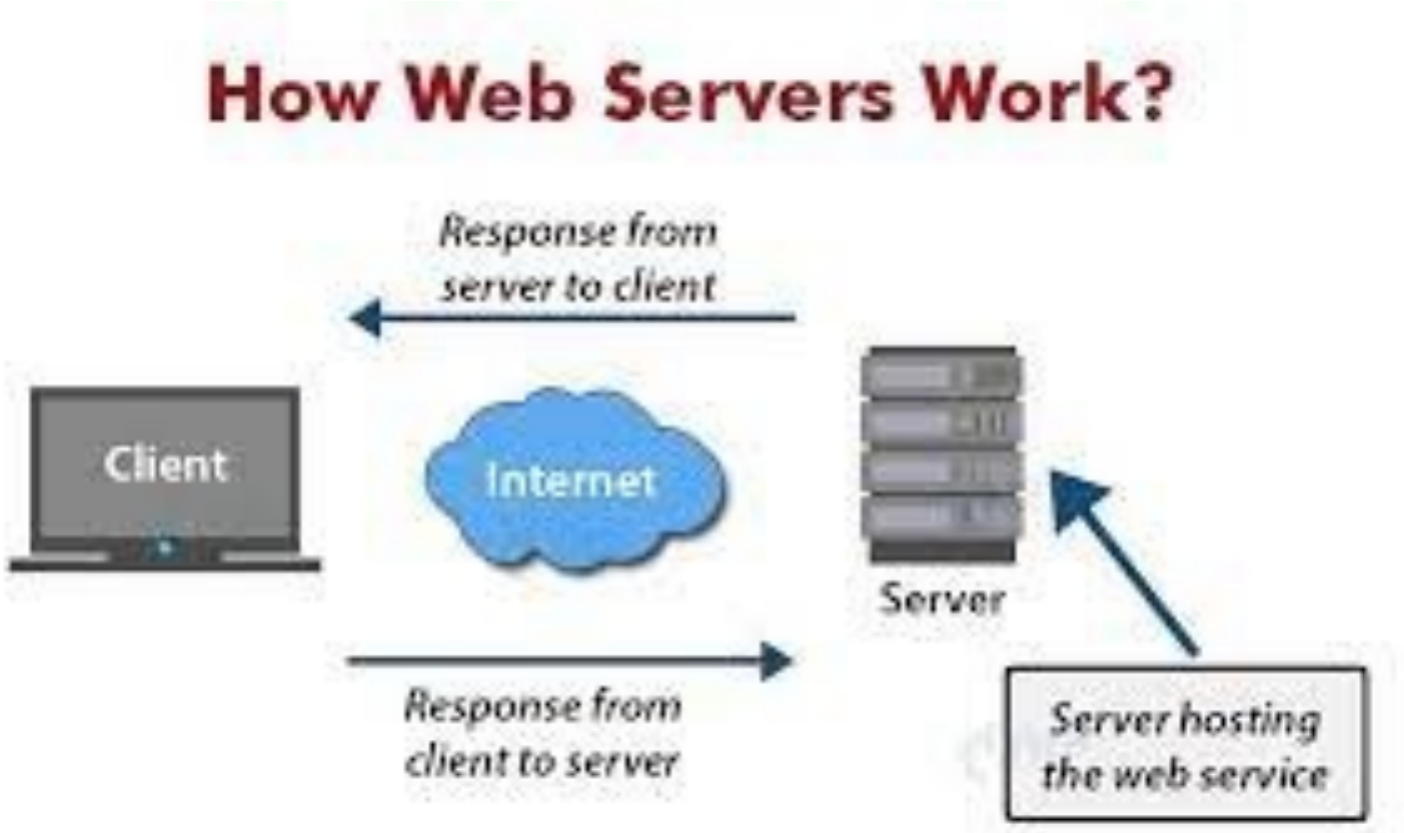
Chapter 4: Application Architecture

- DHRUMIL PAREKH
- SAYALI SANT



WEB SERVICE ARCHITECTURES

The web services architecture is based on interactions between three components: a service provider, a service requester, and an optional service registry.

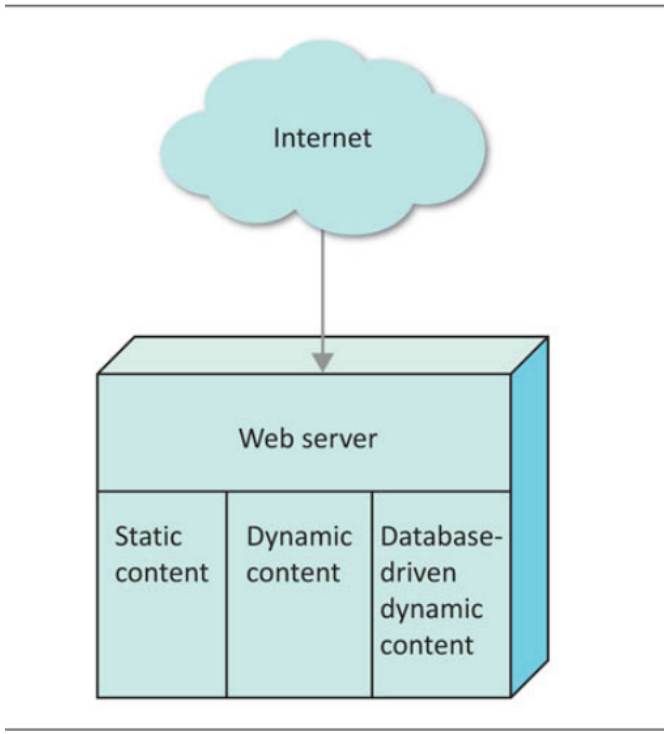
TYPES OF WEB SERVICE ARCHITECTURE

1. Single Machine Web Server

The machine runs software that speaks the HTTP protocol, receiving requests, processing them, generating a result, and sending the reply. Many typical small web sites and web-based applications use this architecture

The web server generates web pages from three different sources:

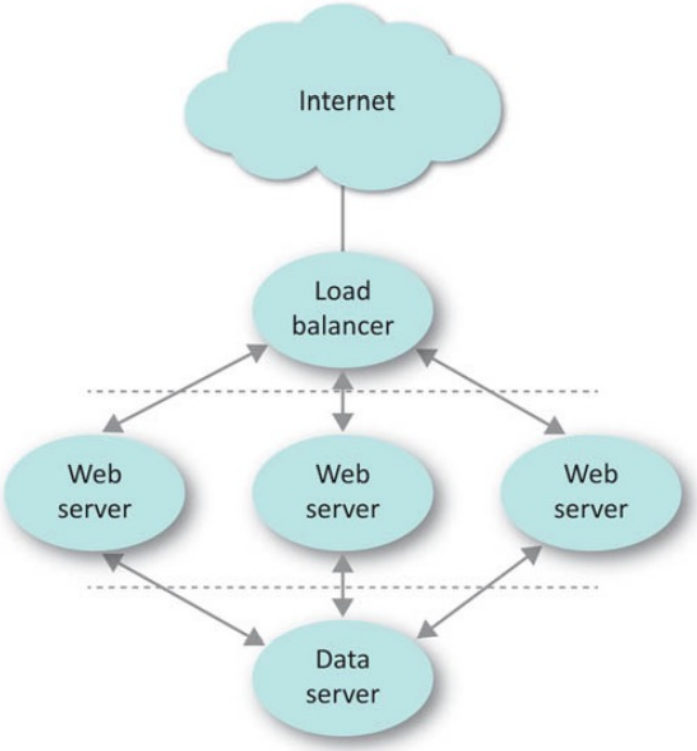
- Static Content
- Dynamic Content
- Database-Driven dynamic content



TYPES OF WEB SERVICE ARCHITECTURE

2. Three-Tier Web Service

The three-tier web service is a pattern built from three layers: the load balancer layer, the web server layer, and the data service layer. The web servers all rely on a common backend data server, often an SQL database. Requests enter the system by going to the load balancer. The load balancer picks one of the machines in the middle layer and relays the request to that web server. The web server processes the request, possibly querying the database to aid it in doing so. The reply is generated and sent back via the load balancer.

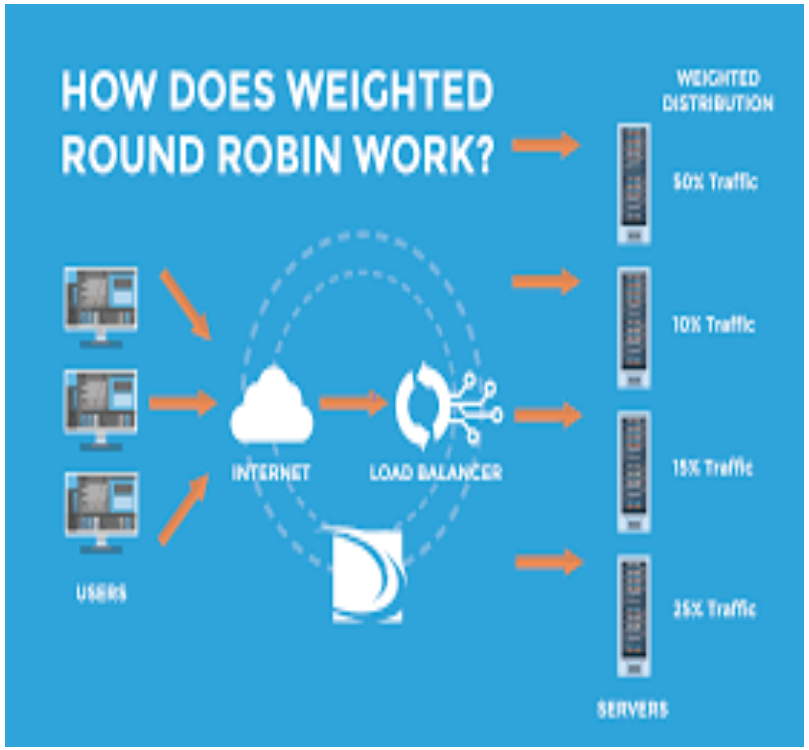


LOAD BALANCER TYPES

A load balancer works by receiving requests and forwarding them to one of many replicas—that is, web servers that are configured such that they can all service the same URLs.

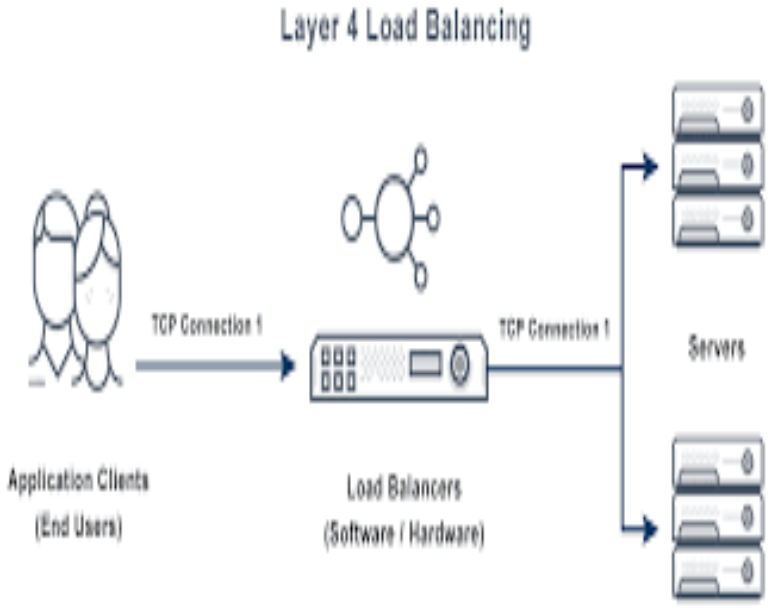
1. DNS Round Robin

This works by listing the IP addresses of all replicas in the DNS entry for the name of the web server. Web browsers will receive all the IP addresses but will randomly pick one of them to try first. Thus, when a multitude of web browsers visit the site, the load will be distributed almost evenly among the replicas.



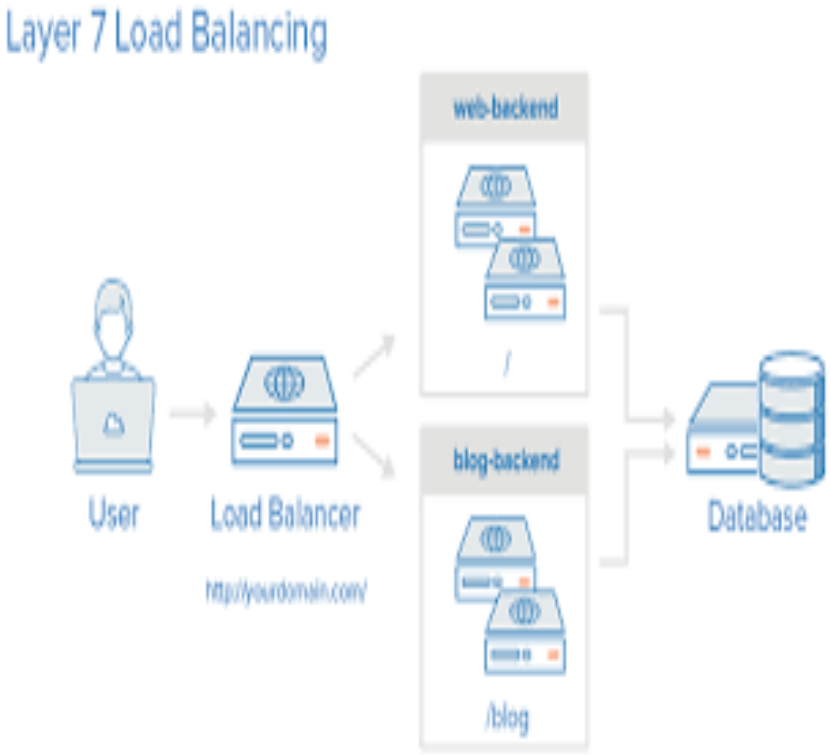
2. Layer 3 and Layer 4 Load balancers

L3 and L4 load balancers receive each TCP session and redirect it to one of the replicas. Every packet of the session goes first through the load balancer and then to the replica. The reply packets from the replica go back through the load balancer. The names come from the ISO protocol stack definitions: Layer 3 is the network (IP) layer; Layer 4 is the session (TCP) layer. L3 load balancers track TCP sessions based on source and destination IP addresses.



Layer 7 Load Balancer

L7 load balancers work similarly to L3/L4 load balancers but make decisions based on what can be seen by peering into the application layer (Layer 7) of the protocol stack. They can examine what is inside the HTTP protocol itself (cookies, headers, URLs, and so on) and make decisions based on what was found. As a result they offer a richer mix of features than the previous load balancers. For example, the L7 load balancer can check whether a cookie has been set and send traffic to a different set of servers based on that criterion. This is how some companies handle logged-in users differently.



Load Balancing Methods

1. Round Robin

The machines are rotated in a loop. If there are three replicas, the rotation would look something like A-B-C-A-B-C. Down machines are skipped.

2. Weighted RR

This scheme is similar to RR but gives more queries to the backends with more capacity. Usually a manually configured weight is assigned to each backend. For example, if there are three backends, two of equal capacity but a third that is huge and can handle twice as much traffic, the rotation would be A-C-B-C.

Load Balancing Method

3. Least Loaded (LL)

The load balancer receives information from each backend indicating how loaded it is. Incoming requests always go to the least loaded backend

4.Least Loaded with Slow Start

This scheme is similar to LL, but when a new backend comes online it is not immediately flooded with queries. Instead, it starts receiving a low rate of traffic that slowly builds until it is receiving an appropriate amount of traffic.

Load Balancing With Shared State

Suppose one HTTP request generates some information that is needed by the next HTTP request. A single web server can store that information locally so that it is available when the second HTTP request arrives. But what if the load balancer sends the next HTTP request to a different backend? It doesn't have that information (state). This can cause confusion.

Strategies to deal with this situation:

1. Stickiness

Load balancers have a feature called stickiness, which means if a user's previous HTTP request went to a particular backend, the next one should go there as well.

2. **Shared State**

In this case the fact that the user has logged in and the user's profile information are stored somewhere that all backends can access. For each HTTP connection, the user's state is fetched from this shared area. With this approach, it doesn't matter if each HTTP request goes to a different machine. The user is not asked to log in every time the backends are switched.

3. **Hybrid**

When a user's state moves from one backend to another, it generally creates a little extra work for the web server. Sometimes this burden is small and tolerable. For some applications, however, it is extremely inconvenient and requires a lot more processing. In that case using both sticky connections and shared state is the best solution.

User Identity

when a user logs into a web application, the web application generates a secret and includes it with the reply. The secret is something generated randomly and given to only that user on that web browser. In the future, whenever that web browser sends an HTTP request to that same web app, it also sends the secret. Because this secret was not sent to any other user, and because the secret is difficult to guess, the web app can trust that this is the same user. This scheme is known as a cookie and the secret is often referred to as a session ID.

Scaling

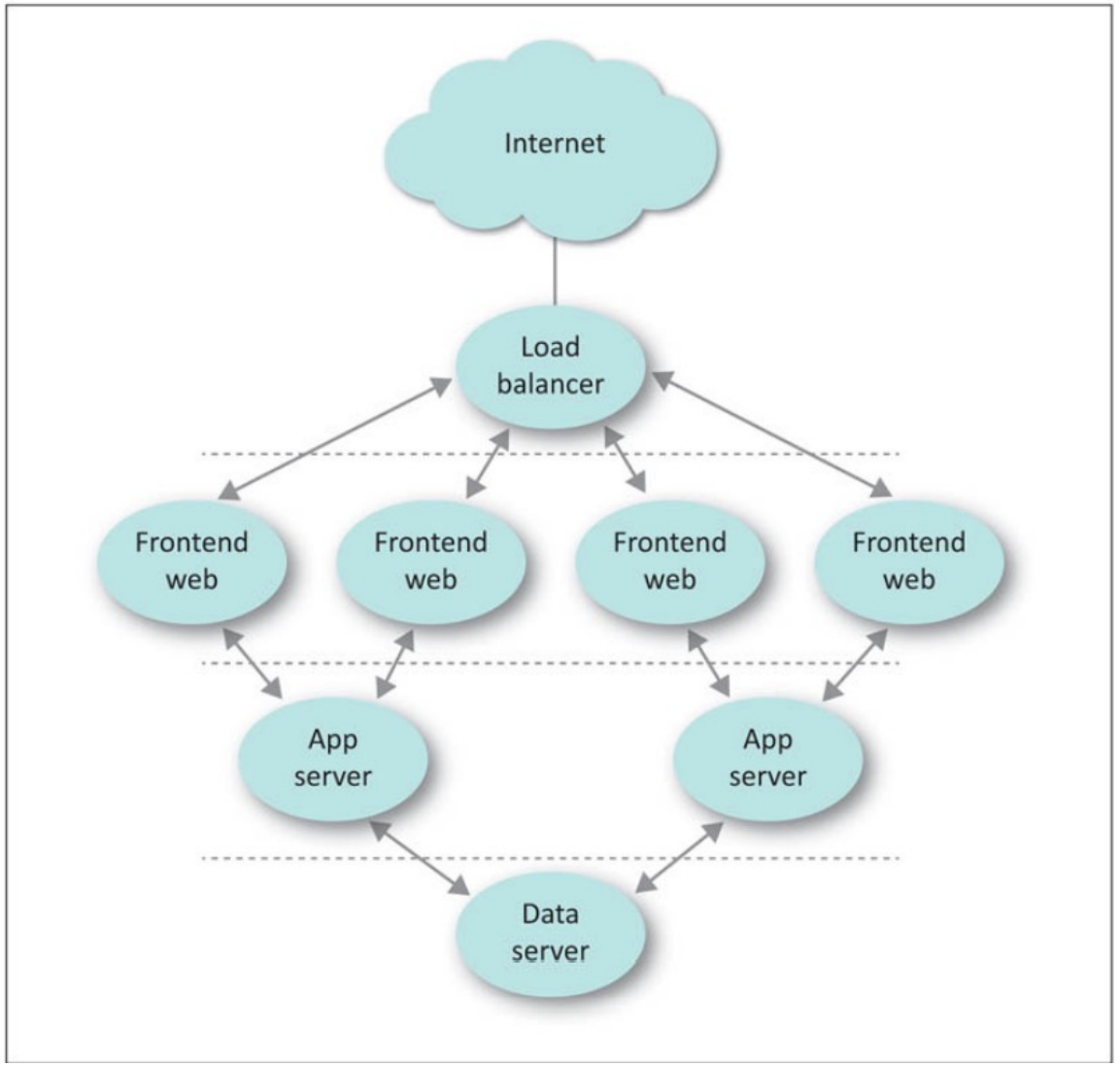
The three-tier web service has many advantages over the single web server. It is more expandable; replicas can be added. If each replica can process 500 queries per second, they can continue to be added until the total required capacity is achieved.

This pattern is also very flexible. This leads to many variations:

1. Replica Groups

2. Dual Load Balancers

3. Multiple Data Stores



Four-Tier Web Service

A four-tier web service is used when there are many individual applications with a common frontend infrastructure. In this pattern, web requests come in as usual to the load balancer, which divides the traffic among the various frontends. The frontends handle interactions with the users, and communicate to the application servers for content. The application servers access shared data sources in the final layer. The difference between the three-tier and four-tier designs is that the application and the web servers run on different machines.

Deep Dive into Architecture

1. **Frontends**

The frontends are responsible for tasks that are common to all applications, thus reducing the complexity of the applications. The frontends handle all cookie processing, session pipelining,, compression, and encryption.

2. **App server**

It is responsible for hosting the application and communicating with frontends and data server for user assistance.

3. **Data Server**

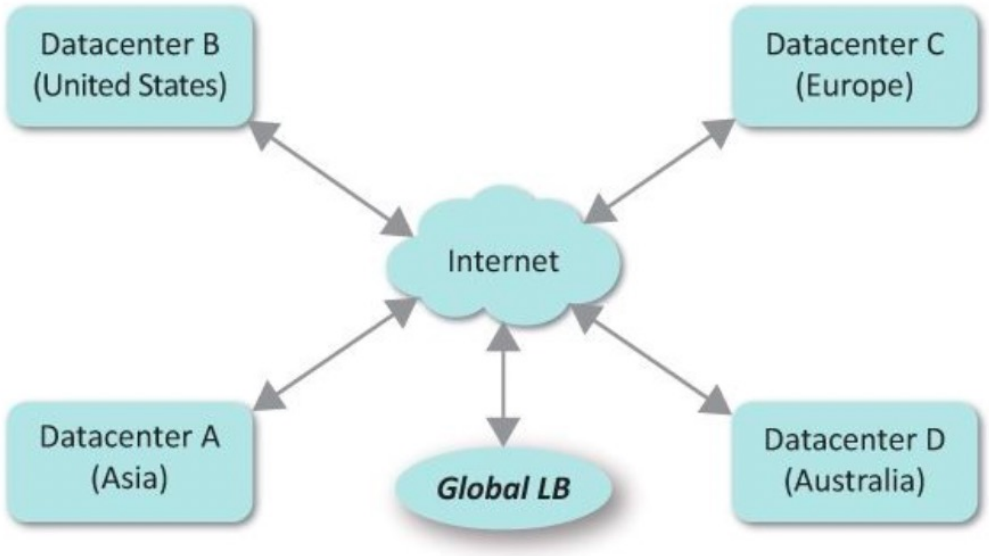
It is responsible for storing the application data and responding to queries whenever a request is needed.

Reverse Proxy Service

A reverse proxy enables one web server to provide content from another web server transparently. The user sees one cohesive web site, even though it is actually made up of a patchwork of applications.

For example, suppose there is a web site that provides users with sports news, weather reports, financial news, an email service, plus a main page:

- www.company.com/ (the main page)
- www.company.com/sports
- www.company.com/weather
- www.company.com/finance
- www.company.com/email



Cloud Scale Service

- Globally Distributed
- The service infrastructure uses one of the previously discussed architectures, such as the four-tier web service, which is then replicated in many places around the world
- It takes time for packets to travel across the internet. The farther the distance, the longer it takes
- We fix this problem by bringing the data and computation closer to the users.

Global Load Balancer

- Global Load Balancer (GLB) is a DNS service that directs traffic to the nearest data center
- A GLB examines the source IP address of the query and returns a different result depending on the geolocation of the source IP address
- A GLB maintains a list of replicas, their locations, and their IP addresses

Global Load Balancing Methods

1. **Nearest:** Strictly selects the nearest datacenter to the requester.

2. Nearest with Limits: The nearest datacenter is selected until that site is full. At that point, the next nearest datacenter is selected.

3. **Nearest by Other Metric:** The best location may be determined not by distance but rather by another metric such as latency or cost

Global Load Balancing with User-Specific Data

When the HTTP request is directed to the nearest datacenter, it will be handled by local load balancers, frontends, and whatever else makes up the service.

This brings up another architectural issue. Suppose the service stores information for a user. What happens if that user's data is stored at some other data center?

GLB works on a DNS level, so it cannot determine the user and hence cannot return the IP of the service and user data.

- Replication of Data

- Frontend communicates with the service to determine the geolocation of the user's data and then accesses the application and its data from the correct datacenter
A private Wide Area Network for WAN is owned by the company which ensures the connection between all the data centers

Internal Backbone - Internet Service Providers

Wherever there is a datacenter, the datacenter will generally connect to many Internet Service Providers (ISPs) in the area

If you do not have a direct connection to a particular ISP, then sending data to users of that ISP involves sending the data through another ISP that connects your ISP and theirs. This ISP in the middle is called a **transit ISP**

The transit ISP charges the other ISPs for the privilege of permitting packets to travel over its network

There are often multiple transit ISPs between you and your customers. The more transits, the slower, less reliable, and more expensive the connections become.

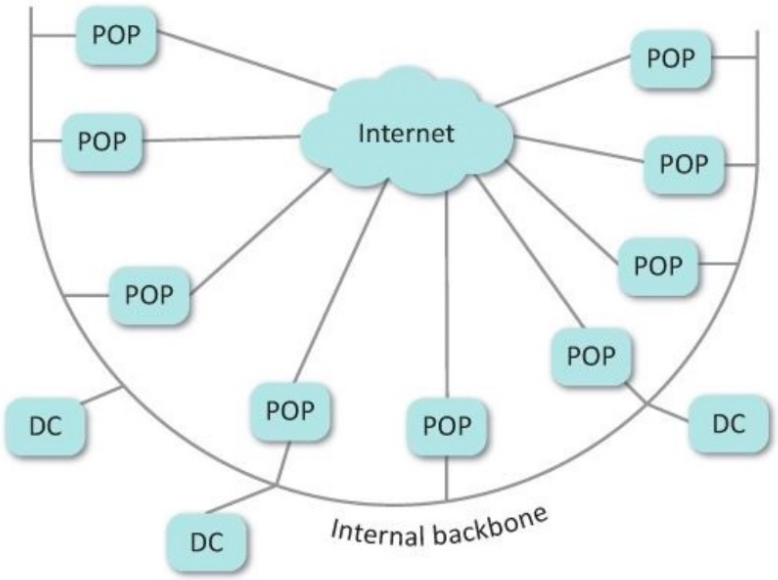
Internal Backbone - Point of Presence (POP)

A **point of presence (POP)** is a small, remote facility used for connection to local ISPs.

It is advantageous to connect to many ISPs but they cannot always connect to your datacenter

Your data center may not be in the state or country they operate in. Since they cannot go to you, you must extend your network to someplace near them.
For example, you might create a POP in Berlin to connect tc many different German ISPs.

A POP is usually a rack of equipment in a colocation center and contains network equipment and connections from various telecom providers



Satellite, Frontends and Content Distribution Servers

A POP plus a small number of computers is called a **satellite**.

The computers are used for frontends and content distribution services.
The **frontends** terminate HTTP connections and proxy to application servers in other datacenters.

Content distribution servers are machines that cache large amounts of content. For example, they may store the 1000 most popular videos being viewed at the time

Thus, an internal network connects datacenters, POPs, and satellites.

How Global Load Balancer handles traffic for Google Maps

Should the GLB direct traffic to the nearest front end or to the nearest datacenter that has application servers for Google Maps?

Path 1: GLB directs the traffic to the nearest frontend

Whether that frontend is in a datacenter or satellite doesn't matter. There's a good chance it won't be in one of the datacenters that host Google Maps, so the frontend will then talk across the private backbone to the nearest datacenter that does host Maps.

This solution will be very fast because Google has total control over its private backbone, which can be engineered to provide the exact latency and bandwidth required.

There are no other customers on the backbone that could hog bandwidth or overload the link

However, every packet sent on the backbone has a cost associated with it, and Google pays that expense.

Should the GLB direct traffic to the nearest front end or to the nearest datacenter that has application servers for Google Maps?

Path 2: GLB directs the traffic to the frontend at the nearest datacenter that hosts Google Maps

If that datacenter is very far away, the query may traverse multiple ISPs to get there, possibly going over oceans. This solution will be rather slow.

These transcontinental links tend to be overloaded and there is no incentive for the ISPs involved to provide stellar performance for someone else's traffic.

Even so, the cost of the transmission is a burden on the ISP, not Google. It may be slow, but someone else is bearing the cost.

Should the GLB direct traffic to the nearest front end or to the nearest datacenter that has application servers for Google Maps?

Solution:

We choose both the paths

- Google wants its service to be fast and responsive to the user. Therefore, traffic related to the user interface (UI) is sent over the first path. This data is HTML and is generally very small.

- The other part of Google Maps is delivering the map tiles—that is, the graphics that make up the maps. The graphics use a lot of bandwidth.
The map tiles can load very slowly, and it would not hinder the users' experience. Therefore, the map tile requests take the slow but inexpensive path.

Thus, users get fast interaction and Google pays less for bandwidth.

Message Bus Architecture

- A message bus is a many-to-many communication mechanism between servers. It is a convenient way to distribute information among different services.
- A message bus is a mechanism whereby servers send messages to “channels” (like a radio channel) and other servers listen to the channels they need
- A server that sends messages is a **publisher** and the receivers are **subscribers**.
- A server can be a publisher or subscriber of a given channel, or it can simply ignore the channel. This permits one-to-many, many-to-many, and many-to-one communication.
- A message bus system is efficient in that clients receive a message only if they are subscribed. Thus network bandwidth and processing are conserved.
- Message bus technology goes by many names, including message queue, queue service, or pubsub service. For example, Amazon provides the Simple Queue Service (SQS), MCollective is described as publish subscribe middleware, and RabbitMQ calls itself a message broker.

Message Bus Designs

- The **message bus master** learns the underlying physical network topology and for each channel determines the shortest path a message needs to follow
- Determining and optimizing the unique topology of each channel separately requires a lot of calculations, especially when there are thousands of channels.
- Some message bus systems require all messages to first go to the master for distribution. Consequently, the master may become a bottleneck. Other systems are more sophisticated and either have multiple masters, one master per channel, or have a master that controls topology but does not require all messages to go through it.
- Channels may be open to anyone, or they may be tightly controlled with ACLs determining who can publish and who can subscribe.
  On some systems, the listeners can send a reply to the message sender. Other systems do not have this feature and instead create a second channel for publishing replies.

Message Bus Reliability

- Message bus systems usually guarantee that every message will be received.
- For example, some message bus systems require subscribers to acknowledge each message received. If the subscriber crashes, when it returns to service it is guaranteed to receive all the unacknowledged messages again; if the acknowledgment did not make it back to the message bus system, the subscriber may receive a second copy of the message.
- Message bus systems do not guarantee that messages will be received in the same order as they were sent
- For this reason, the subscriber must be able to handle messages arriving out of order. The messages usually include timestamps, which help the subscriber detect when messages do arrive out of order.

Service Oriented Architecture

- Service-oriented architecture (SOA) enables large services to be managed more easily.
- With this architecture, each subsystem is a self-contained service, providing its functionality as a consumable service via an API.
- Services communicate with each other using API calls
- A goal of SOAs is to have the services be loosely coupled. This makes it easier to improve and even replace a service.
- Characteristics of SOAs:
  - ✓ Flexibility
  - ✓ Support

Best Practices for Running SOA

- Use the same underlying Remote Procedure Call(RPC) protocol to implement the APIs on all services. This way any tool related to the RPC mechanism is leveraged for all services.

- Have a consistent monitoring mechanism. All services should expose measurements to the monitoring system the same way.

- Use the same techniques with each service as much as possible.
  Use the same load balancing system, management techniques, coding standards, and so on.
  As services move between teams, it will be easier for people to get up to speed if these things are consistent.

- Adopt some form of API governance. When so many APIs are being designed, it becomes important to maintain standards for how they work. These standards often impart knowledge learned through painful failures in the past that the organization does not want to see repeated.

Thank you