CMPSCI 677 Distributed Operating Systems **Spring 2025**

Lecture 21: April 18

Lecturer: Prashant Shenoy *Scribe: Sergei Pogorelov (2025) Ibrahim Hasaan (2024)*

21.1 Web Caching

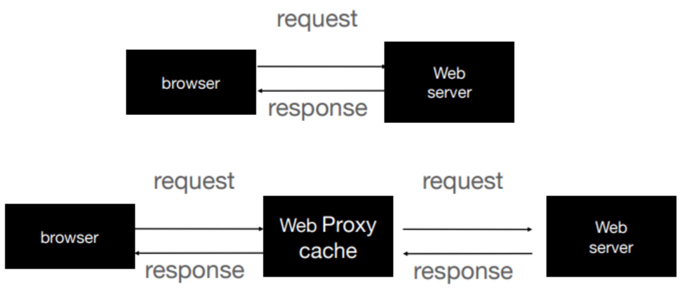

Figure 21.1: Client-Server and Client-Proxy-Server Architectures

Web caching uses a client-proxy-server architecture. Clients send requests to the proxy. Proxies can service the requests directly if they have the resources. If they do not have the resources, they go to the server to get the request processed. In web caching, the proxy provides the service of caching i.e. the proxy caches content from the server and when the client browsers make a request, if the content is already in the cache, it returns the content. This helps in faster responses for the client, if the proxy is near the client, and reduced loads for the server.

Note: Web pages can either be dynamic or static. Dynamic web pages are regenerated for every request and for every user, as opposed to static web pages which can just be pregenerated HTML files which only change when the user changes them. Therefore, dynamic web pages are typically not cached. For this reason, content being referred to in these notes is largely static content (static audio, video, web pages etc).

21.2 Web Proxy Caching

The discussion for this section assumes a collection of proxy caches sitting in between the client and the server. Along with communicating with the server, these proxy caches can also communicate with each other. This mechanism is called “Cooperative Caching”.

Figure 21.2 shows one such scenario where a client sends a request to the web proxy. The web proxy will then look into its local cache for the requested web page. If it is a hit, then it will send the response back. In the case of miss, typically the web proxy will contact server. But in the case of cooperative caching, cache misses can be serviced by asking a nearby/local proxy instead of the server i.e. it will reach out to the near by proxies to get the data. In this case, all the caches act like one big logical cache, so the clients will see union of all the content stored in nearby caches rather than just the content cached in the local proxy. This can make fetching faster than getting data from the server.

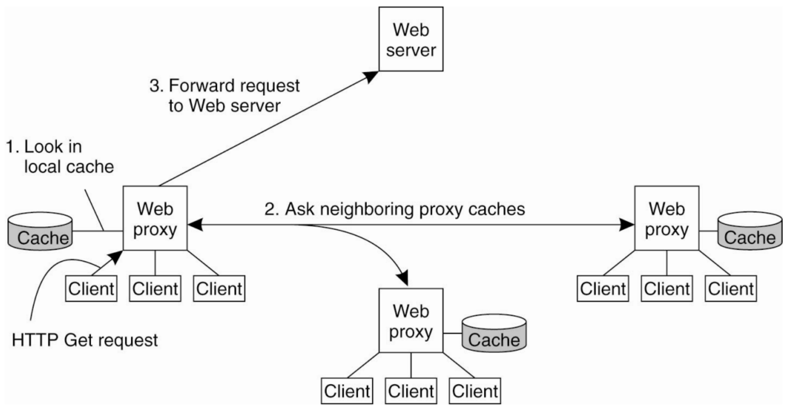

Figure 21.2: Web Proxy Caching

21.3 Consistency Issues

Recall a system has consistency if all the replicas see the same data at the same time. In the context of web caching, this means that when a browser fetches a page, we are guaranteed that the returned page is the most recent version. When designing consistency techniques for web caching, it's important to keep in mind that web pages can vary significantly in terms of how frequently they are updated and accessed:

- Some objects are static, meaning once created, they never change.
- Other web pages are dynamic and change frequently (e.g., news websites with breaking news sections that update every few minutes).
- Update frequency can range from a few minutes to a few weeks to never.
- Access frequency (popularity) also varies, with some pages receiving millions of users per day, while others get very little access.

There are 2 approaches for maintaining consistency - pull-based and push-based.

21.3.1 Push based Approach

This approach relies on the server, i.e. it is the responsibility of the server to ensure that the content stored at the proxy caches is up to date. For every web page stored at the server, the server keeps a table of all proxies that have a cached copy of that web page. Whenever a page is updated, the server looks at the table, finds all the proxies that have a copy of the web page and notifies each of them that the page has been updated.

This notification can be of 2 types:

1. **Invalidate** - inform the proxy that the web page has changed (so it can discard the page from its cache).
2. **Update** - send the new version of the page. This tells the proxy that the page was changed and that it should replace the old version (in its cache) with the new one.

Question: When to use invalidate and when to use the update message?
Answer: This depends on the read and update frequency of the content.

- **Invalidate**: Use invalidate for less popular pages (low access frequency). Since fewer clients will request the updated page, this avoids wasting bandwidth sending updates to proxies unnecessarily.
- **Update**: Use updates for frequently accessed pages, as most proxies will request the updated page after receiving the invalidate message. Sending the updated content directly avoids redundant requests.

Advantages:

- Provides relatively **tight consistency guarantees**.
- Proxies can remain **passive**; the server manages consistency.

Disadvantages:

- The server needs to maintain substantial **state**, including information about which proxies have cached specific content.
- Requires the server to remain **stateful**, which increases complexity.
- **Failure recovery** becomes more difficult due to the state being lost when the server crashes.

Question: Given that the cache size at a proxy is limited, what happens when old items are removed due to cache replacement? How does the server know that content has been evicted from the proxy's cache?

Answer: When the cache size is limited, proxies must implement a **cache replacement policy**, such as **Least Recently Used (LRU)**, where new items are brought in and old or less frequently used items are removed.

The server, however, needs to track whether cached content is still present in the proxy to avoid sending unnecessary updates or invalidation messages. To inform the server about evictions:

1. The proxy could notify the server whenever an item is removed from its cache.
2. If such notifications are not implemented, the server will not be aware of specific evictions and might send wasteful update or invalidation messages unnecessarily.

This additional communication adds overhead to the system but ensures the server has an accurate understanding of the proxy's cache state.

Question: While the invalidate message is in transit, the cache is already stale. How to deal with this?
Answer: We cannot deal with this issue because even if the server instantly sends the message there is still a speed of light propagation delay for the message to reach the proxy. For this period of time, the proxy is going to have stale code. Strict consistency in practice is hard to achieve.

21.3.2 Pull based Approach

In a **pull-based approach**, the **proxy** is responsible for maintaining cache consistency by periodically querying the server to check whether the cached page is still valid. This is accomplished using **HTTP**

conditional GET messages (e.g., "If-modified-since"). Server will return modified web page only if it has changed.

When to poll for web pages?
Depends on the frequency of updates. For web pages which change very rarely, frequent polling is extremely wasteful. There are no such wasteful messages in the push based approach. If the page changes much more frequently than the polling frequency, then the proxy might cache outdated content for longer times. Thus, polling frequency should be decided based on the update frequency of web page. There are two ways to do this:

1. Server can assign an expiration time, TTL(time-to-live). This time is an estimation of next possible changes on the web page. Server can estimate it based on the past web page update frequency history. It is likely that web page might change after TTL so poll after TTL expires.
2. Proxy has intelligence to dynamically figure out the polling times. Poll duration is varied based on the observed web page updates. Dynamically change polling frequency to understand the average rate at which web page is changing.

Adapting polling frequency dynamically:

- Initially poll frequently to detect how often the page changes.
- Over time, adjust polling to match observed update frequency (increase intervals for static objects, poll more frequently for dynamic objects).

Question: In a pull-based approach, do we have distinct message types like in the push-based approach, where there are separate notify messages for invalidation and updates? Or is it handled differently?

Answer: In the pull-based approach, it works a bit differently. You are always using a **conditional GET**, which functions as both a notify and an update:

- If the content has **not changed**, the conditional GET returns a `null`, essentially serving as a **notify** message, informing the proxy that nothing has changed.
- If the content **has changed**, the condition in the GET request evaluates to `true`. In this case, the updated content is fetched, making it equivalent to an **update**.

Thus, instead of separate message types as in the push-based approach, the conditional GET in the pull-based approach combines both functionalities. Its response behaves like a notify or an update depending on whether the condition is `true` or `false`.

Question: Why not just poll when you get the request?
Answer: We could do this, this is the only way we get strong consistency because when a request comes and it's a cache hit. But we don't know if it's consistent, so we can do an If-modified-since request to check if the cache still consistent.

Question: What is the downside of the approach mentioned in the above question?
Answer: Increase in latency. The reason we are caching the content at the proxy is that proxies are close to the client and they can deliver content faster. However, before delivering that content every time if you have to go to the server to check, it introduces another round trip time delay. You may as well go to the server and fetch the content directly.

Question: In the pull-based approach, does the server have additional overhead because of the additional If-modified-since http messages?

Answer: Yes, there is additional overhead because these checks may need to be made frequently to maintain consistency.

Question: Does DNS use a pull-based approach?
Answer: DNS is a pull-based protocol in the sense that it translates hostnames to IP addresses, and the browser sends it a hostname for which the DNS returns the IP address. DNS also uses caching, for which it just uses TTL values (since IP addresses don't change that frequently).

Question: Why can't we do a hybrid of both approaches (Pull-based and Push-based)? Why can't the developer then decide, or the proxy decide dynamically?
Answer: The server has to support a push-based approach. For example, if the server was just a standard http server then it won't have any push-based functionality. If the server does, then you could do a hybrid approach.

Question: Do we need to worry about clock synchronization?
Answer: Not too much, assume clocks use NTP accuracy of 10 milliseconds. As web pages won't change every 10 milliseconds (usually changed in days or weeks), we need not worry about synchronization.

Question: Polling is to keep the content in the cache consistent with the server, but how does the proxy know whether to keep the cached content at all? Maybe nobody is interested in it.
Answer: To understand when a proxy should maintain cache and when not to it has to track some statistics (like the rate a page is requested at). If the request rate is very low, the proxy can decide the page is no longer popular and evict it. On the other hand, if the request rate for a page is high proxy will decide to keep it consistent.

Question: Pull-based approach does not have persistent HTTP connections, is that a limitation of this approach? And does server push have persistent connection?
Answer: Neither approach requires persistent connections. If polling frequency is less does not make sense to keep persistent connections - wasting lot of resources. Only makes sense to do this when content is changing frequently.

Advantages and Disadvantages
Pull based approach gives weaker consistency guarantees. Updates of a web page at server, might not be immediately reflected at proxies. Latency to synchronize the content might overtake the benefits of proxies. There is a higher overhead than the server push approach, and there could be more pulls than updates. Some advantages are that the pull based approach can be implemented using HTTP (server remains stateless). This approach is also resilient to both server and proxy failures.

21.3.3 A Hybrid Approach: Leases

Hybrid approach based on both push and pull. Figure 21.3 illustrates how it works.

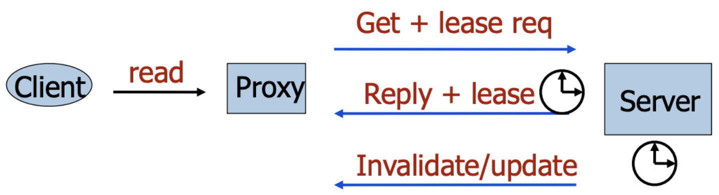

Figure 21.3: Lease

Lease is a contract between two entities (here the server and the proxy). It specifies the duration of time the server agrees to notify the proxy about any updates on the web page. Updates are no longer sent by the server after lease expiration, and the server will delete the state. Proxy can renew the lease. If the page is unpopular, proxy can decide not to renew the lease for that page.

Lease is a more limited form of server push - performing push for the duration of the lease only. We get advantages of push and do not have to keep state indefinitely. If the lease duration is zero, it degenerates to polling (pull). If duration is infinite, it is the same as push-based and makes the server stateful. Duration in between zero and infinite is a combination of both pull and push.

Tuning the lease duration:

- Long leases: Suitable for **frequently accessed** content or objects that change infrequently.
- Short leases: Suitable for **rarely accessed** content or when the server is highly loaded, as this reduces the cost of tracking state at the server.

Tight consistency guarantees when lease is active.

21.3.4 Policies for Leases Duration

Lease duration is an important parameter. There are three policies for lease duration:

1. **Age-based:** Based on frequency of changes to the object. The age is the time since last update. Assign longer leases for more frequently updated pages.
2. **Renewal-frequency based:** Based on frequency of access requests from clients. Popular objects get longer leases.
3. **Server load based:** If the load on the server is high, we should use shorter lease duration. This will remove the burden of storing proxy state on server side.

Question: What is age-based lease?
Answer: First, what does age mean here? The age of a file or web content is the time since it was modified last i.e. time now - time file was last modified. If a file has old age it means it hasn't been modified and a younger file means they've been modified recently. We can use age to decide how long a lease should be.

Question: If the file is modified more frequently, the age will be lower so the lease time will be longer?
Answer: In the slide it says to give longer leases to the objects with larger lifetimes, but you can do exactly the opposite of this depends if you want to reduce the workload on the server or not. Giving long leases for frequently updated objects increases the server load.

Question: If a proxy evicts an object from the lease, can the lease be canceled? **Answer:** Original lease mechanism does not have any cancellation mechanism, but you can add one. Ideally, if you have an active lease you should not evict an object but if you do for any reason, there has to be either a way to cancel the lease or you'll get some wasted notifications from the server.

Question: Does each proxy have its own lease?
Answer: Not only does each proxy have a lease, there is a lease for each web page at a proxy. If a proxy caches 100 different web pages, each of them will have a different lease. It is not a lease per proxy, it is a lease per web page at a proxy. Different proxies will have different leases for the same page.

Question: Is it the proxy's responsibility to keep track of the lease or the server's responsibility?
Answer: Both. Server has to keep track of all active leases and send notifications whenever the web page

changes for every active lease on that web page. Proxy has to track lease as well. If the lease expires, proxy decides whether to renew it or not.

Question: Do you need a separate monitoring framework to keep track of popularity of content?
Answer: Server by itself will not know how popular the content is. Server can know the age and server load. Proxies can track popularity of the web page and report stats to the server. Monitoring framework can be added to the proxy system, it is not a part of the lease approach.

21.3.5 Cooperative Caching

Recall, in cooperative caching a collection of proxies cooperate with each other to service client requests. These proxies can be arranged in different structures. In "Hierarchical Proxy Caching", the proxies are arranged in a hierarchy, a tree-like structure, where the server is the root and the rest of the nodes are caches.

Figure 21.4 shows an example of this. The client sends a request to one of the proxies. If it is a cache miss, the proxy will send requests to its peers (red arrows) and parent using ICP (Internet Cache Protocol) messages. If none of the peers have it, they will send back the non-availability response to the proxy (green arrows). The proxy will then forward the HTTP request to its parent and the whole process will recurse until a cache hit occurs or until the server is reached. The data will then be sent back as response - and will flow down the hierarchy, back to the client.

Question: Why are we using ICP instead of HTTP?
Answer: ICP is designed specifically for caching and cache consistency, whereas HTTP is not. ICP is basically just a way to ask other nodes if they have some content and fetch the content if they do, so its not very different to HTTP in that sense.

Question: What if the proxies have different versions of the file?
Answer: Here, the assumption is that if any one proxy has the content, then that content can be sent. To maintain consistency, whenever a proxy knows that the content has changed it can inform other proxies of the latest version.

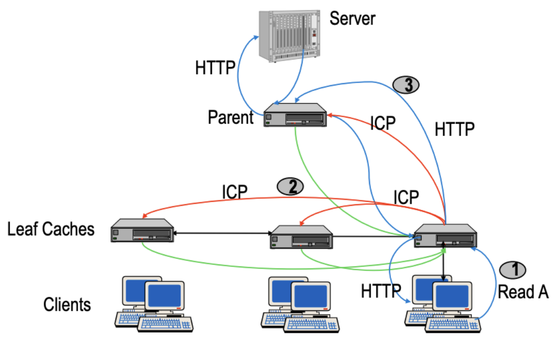

Figure 21.4: Hierarchical Proxy Caching

This works well when a nearby proxy actually has the content. If there is a global miss - no one in the hierarchy has the content, latency will increase significantly. Clearly, there is a lot of messaging overhead. Also, browser has to wait for longer times in the case of cache miss on the proxy. This will affect performance. Latency could increase - it may have been faster to just directly request from the server in the event of a cache miss on the proxy.

To address this problem, we'll look at a different approach for doing this. Every time we send a request up the chain, it adds more hops which increases the overhead and thus the latency, so firstly we'll remove the hierarchy. This gives us a flat structure where every proxy directly communicates with all other proxies. If the content for a request is in a nearby cache, it is fetched. Otherwise, the content is fetched directly from the server.

Moreover, we want to reduce the overhead of querying other proxies to see if they have a page or not. To achieve this, every proxy keeps track of what is stored in its nearby caches. Now, whenever a request comes in, if the proxy doesn't have the data in its cache, it looks in the table to see if any nearby cache has it. And likewise, if there is no entry in the table for a request i.e. none of the nearby caches have the content, it gets it from the server and adds it to the table.

Using this approach, lookup is local. A hit is at most 2 hops and a miss is also at most 2 hops (as opposed to 1 hop in the other method). Every time the proxy fetches or deletes a page, it updates the table for all the nodes. Every proxy keeps a global table that must be kept consistent. There is an additional overhead for keeping this consistent, but the performance is better.

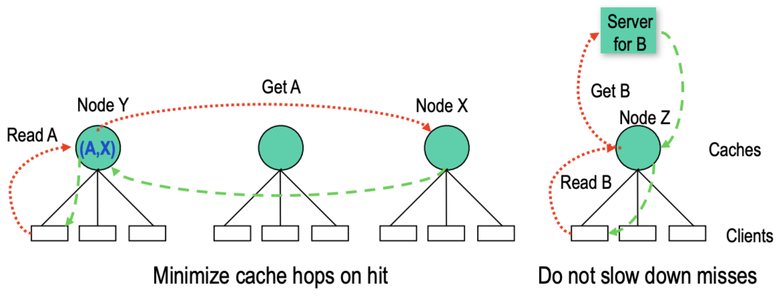

Figure 21.5: Locating and Accessing Data in the Flattened Network

Question: Do we force the miss by enforcing that it goes to at most two hops?
Answer: Yes, two is the most we will have because there is no re-forwarding. If a request comes in from another proxy, we will not check if some other proxy has it. So, a request from a client and one from another proxy are treated differently.

Question: Is this approach going to be more efficient than the previous one? Because we have to inform all other caches after every cache update.
Answer: There are two things to keep in mind. Firstly, the caches are not going to change that frequently. Secondly, the purpose of having proxies is that the latency times for clients are reduced. Updates can happen in the background, so they don't add to clients' latency.

Question: Why could we not use Hierarchical Proxy Caching if there are only popular web pages?
Answer: We cannot assume users will only request for popular pages they can ask for any page.

Question: In the server push approach or lease approach, can you use multicast to notify all proxies?
Answer: The notifications can be sent as either n unicast messages, one to each proxy that has the content, or a single multicast message, where all proxies are listening, so that is a more efficient way of sending messages. That is independent of whether a lease is being used or not.

Follow-up Question: If you do multicast, don't you require a lease?
Answer: The reason a lease is required is if you don't have a list of which proxies to notify and send an update to every proxy in the system, maybe only 10 out of 10000 proxies have the content. You have now

wasted messages by sending message to 10000 proxies. Even though it is a multicast message, you are still using network resources to send it. If you want to multicast, you want to send it only to the proxy group that has the content and so you need to track that.

Question: If you want to do multicast, how do you identify proxies that have the content?
Answer: You would have to construct a multicast group for every web page. When a proxy caches that content, you put that proxy in that group. When the content is removed from the proxy, you remove the proxy from the group.

Question: If you have a hierarchical proxy caching system, can different proxies use different consistency mechanisms - some push some pull?
Answer: That would be a problem. Since caches are interacting with each other, it is better to use uniform consistency mechanism.

21.4 Edge Computing

It is the evolution of proxy servers into a more general approach where servers are deployed at the edge of the network and they provide a service. These servers can provide more than just caching services. Applications can be run on these servers. Edge computing is a paradigm where applications run on servers located at the edge of the network. Benefits include lower network latency than remote cloud servers, higher bandwidth and can tolerate network or cloud failures.

Cloud computing platforms are treating edge computing as an extension of cloud computing where cloud resources are being deployed closer to users rather than in distant data centers.

21.4.1 Edge Computing Origins

Edge computing evolved simultaneously from mobile computing and web caching.

Content Delivery Networks - As web caching became popular, several commercial providers offered proxy caching as a service - they deployed proxy caches in many different networks. If you were an operator of a web app, you could become a customer and they could cache your content and deliver to your customers, for a small fee. These companies deployed CDNs - large network of proxy caches deployed all over the world. You could offload your content to these proxy caches and deliver to end users at low latency. This network of caches was an early form edge computing.

Mobile Computing - Early mobile devices were resource and energy constrained. Not advisable to do heavy computations on these devices. One approach was to put servers near the edge of the wireless network. Computationally intensive tasks were offloaded to the edge server. This was called computation offloading - offloading work from one device (mobile device) to another (edge server). This approach was also an early form of edge computing by offloading computation at low latency.

21.4.2 Content Delivery Networks (CDNs)

Global network of edge proxies that provide caching services among other services to deliver web content. Useful to deliver rich content like images or video content which can increase the load on server significantly as its better to cache such content to reduce load on server. Commercial CDNs deploy many these servers in many different networks. Servers are deployed as clusters of different sizes depending on the demand.

Content providers are customers of the CDN service. They decide what content to cache and it is their responsibility to maintain consistency.

Users access website normally, the content is fetched by the browser from CDN cache.

21.4.2.1 CDN Request Processing

Figure 21.6 shows how a request is routed from a client to a CDN server. The client is going to go the server and make a request for an html page (this is not cached, so it comes from the origin server). Within the page, there may be content; the URL for that content will point to one of the CDN servers.

Now, the CDN has to decide which of its caches will give the content. This is done through a smart DNS lookup. Recall DNS ("Domain Name Service") is a service which takes the url and gives the IP address. The browser makes a connection to this returned IP address.

In a smart DNS lookup, the DNS Server will check where the client is located and return the IP address of the nearest cache.

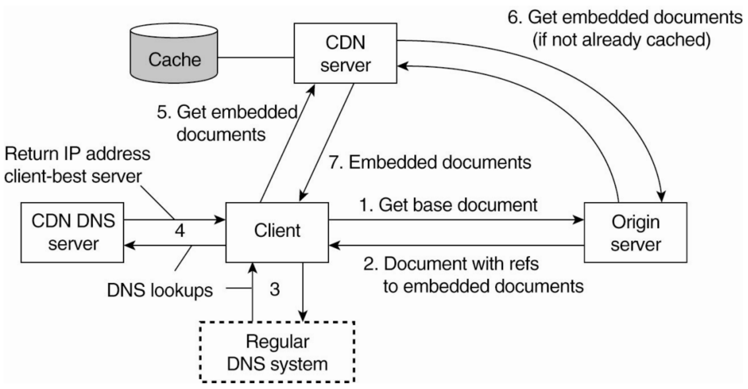
Cache
CDN server
6. Get embedded documents (if not already cached)
5. Get embedded documents
Return IP address client-best server
7. Embedded documents
1. Get base document
CDN DNS server
4
Client
Origin server
2. Document with refs to embedded documents
DNS lookups
3
Regular DNS system

Figure 21.6: CDN Request Processing

21.4.2.2 CDN Request Routing

When a CDN gets a request, it needs to send to the nearest cache (to get the lowest latency), so how does it decide which proxy has to serve that request? They have large load balancers that look at incoming requests and send it to different proxies. Typical CDNs have 2 level load balancer:

1. Global level - Which cluster to send the request to. This is done using DNS (as described above).
2. Local level - Once the request is mapped to a cluster, which server in the cluster will serve the request? (When a request is routed to a cache, it will usually not be a single cache, but a cluster of caches).

Question: If it's a local load balancer does it use concepts like least loaded, round robin?
Answer: Yes, that is exactly what the local load balancer does. As long as content is replicated, you send it to any of the caches; if its not you have to look at the url and only send it to the subset of them that have it.

Question: Do edge proxies use cooperative caching?
Answer: In this case, no need to do cooperative caching. Essentially replicating content and local load balancing will ensure that the server you are getting mapped to has a copy of the content.

Question: Are DNS and CDN independent services?
Answer: In this case, the DNS is run by the CDN server. Your browser will send a request to your local DNS. That DNS server will send that request to another server responsible for the domain you are going after. For example cnn.com/newsvideo.mp4. Cnn would, in this case, have the CDN run its DNS service for it. So, the request goes to CDN where all of this happens i.e deciding the closest server. DNS is indeed separate from CDNs but some DNS servers in this case are going to be run by CDN and those are the servers for domains that it is actually caching content for.

Question: Is there criteria apart from geographical proximity that is used to do load balancing?
Answer: This is the case. Proximity is not the only criterion, there will be a lot more sophistication to handle overload etc. For example - fault tolerance has to be built in.

CDNs have evolved from simple caches to running entire applications at the edge. Figure 21.7 shows CDN hosting web apps. Dynamic content is not cacheable so caches are less useful for CDNs. So CDNs allow running applications on edge server at low latency.

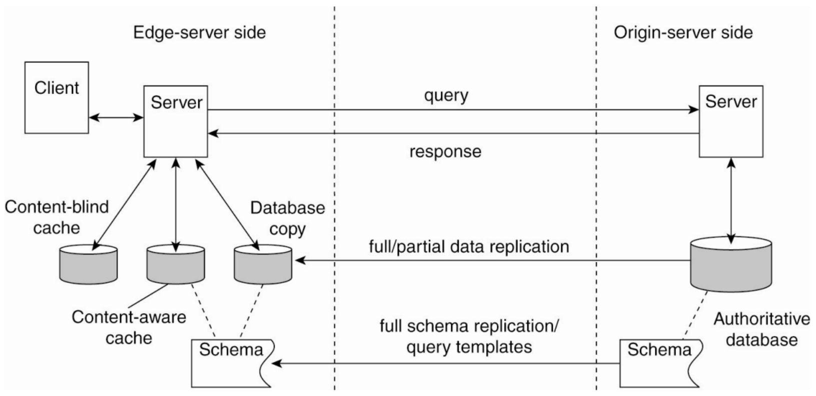

Figure 21.7: CDN Hosting web apps

21.4.3 Mobile Edge Computing

Allows mobile devices to offload compute-intensive tasks to edge servers. Use cases are mobile AR/VR where the mobile device had to process graphics heavy content that drained battery life faster and heavy duty computation was required. Since users are interacting with the system, low latency is very important. Edge servers provided both compute power as well as low latency.

Mobile devices today are much more capable and need to offload from smartphones has reduced. Other devices today that are not as capable such as headsets still use mobile edge computing for offloading compute intensive tasks.