CMPSCI 677 Distributed & Operating Systems **Spring 2022**

Lecture 21: April 13

Lecturer: Prashant Shenoy *Scribe:* ***Mikayla Timm (2019), Saloni Chalkapurkar(2022)***

21.1 Web Caching Continued from the previous lecture

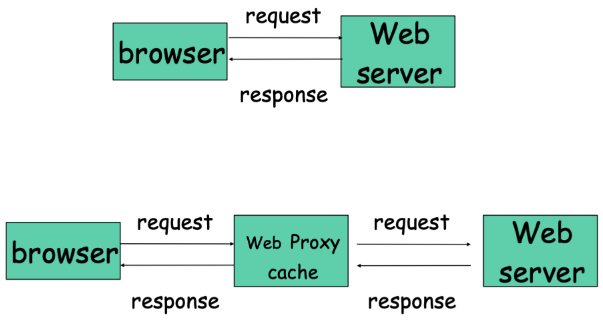

Figure 21.1: Client-Server and Client-Proxy-Server Architectures

Web caching uses 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, the content is returned.

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 it's 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 nearby/local proxy instead of the server i.e. it will reach out to the near by proxies to get the data. This can make fetching faster than getting data from server. Caches are cooperating with each other and clients see union of all the content stored in nearby caches rather than just the content cached in the local proxy.

21.3 Consistency Issues

What can be done if content in the proxy becomes stale? When content is updated at the server, the proxy can no longer serve the content to the users because they will see an outdated version of the webpage.

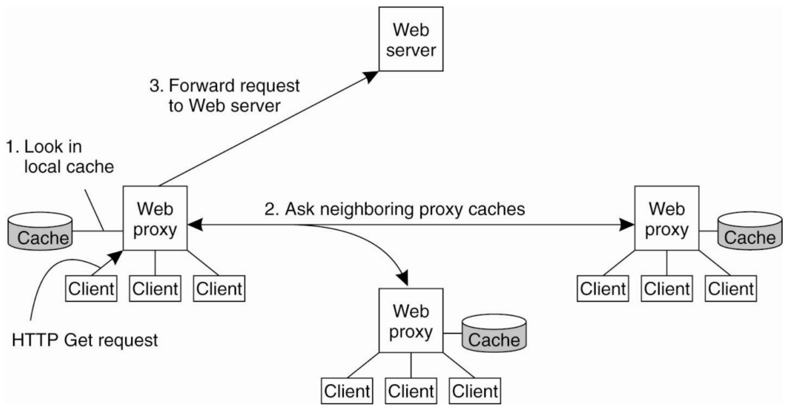

Figure 21.2: Web Proxy Caching

Web pages tend to change with time. Update frequency of web content varies from page to page. When a browser fetches a page from the server, we are guaranteed that the returned page is the most recent version. While using proxies, we need to ensure the consistency of cache web pages. How can the proxy maintain consistency of web content when different webpages get updated at different frequencies? There are 2 approaches for this - pull based and push based. Regardless of the approach, the proxy is notified either by sending an invalidate message or by sending the new updatedversion of the page.

21.3.1 Push based Approach

This approach relies on the server. It is the responsibility of the server to ensure that all the content stored at various proxy caches are never out of date. For every webpage stored at the server, the server keeps a table that lists all proxies that have a cached copy of that webpage. If a page is updated, server looks at the table and finds all the proxies that have a copy of the page and notifies the proxies that the page is updated. This notification can be of 2 types -

1. **invalidate** - simply inform the proxy that the webpage has changed and discard it from the cache.

2. **update** - send the new version of the page which automatically tells the proxy that the page was changed and the new page can be put in the cache and delete the old version.

When to use invalidate and when to use the update message?
Invalidate is a short message which simply notifies whereas update is large message which send the entire content of the updated page. If the content is popular at proxy, it makes more sense to update as it will take time when proxy has to fetch the page from the server again when a subsequent request for it is made. If the content is not popular, it makes more sense to send the invalidate message as update might waste bandwidth by sending the page to the proxy but the proxy has no request for the webpage subsequently.

Push-based approach provides tight consistency guarantees. Proxies can also be passive, as the server does all the work in this approach. Server becomes stateful - for every webpage it keeps track of proxies. One disadvantage is that, since HTTP is stateless, you need mechanisms beyond HTTP. Another disadvantage is that the server needs to keep track of proxies forever.

21.3.2 Pull based Approach

Relies on the proxy. Proxy is responsible to maintain consistency. It polls the server periodically to check if the previously cached page has changed. Polling is performed using conditional GET (if-modified-since HTTP messages). That is if the web page has changed since the timestamp 't', GET me the updated web page. Will return modified webpage only if it has changed.

When to poll for webpages?
Depends on the frequency of updation. 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 values). 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 webpage 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 average rate at which webpage is changing.

Question: Can this change be done like using congestion control techniques like AIMD(Additive Increase Multiplicative Decrease) to figure out congestion window using network congestion?
Answer: This idea was directly inspired by congestion control techniques where you are probing in that case the network to figure out the congestion window, here you are probing the server.

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

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 sync the content might overtake the benefits of proxies. There is a higher overhead than 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.

Question: Pull based approach does not have persistent HTTP connections is that a limitation of this approach and of server push has 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.

21.3.3 A Hybrid Approach: Leases

Hybrid approach based on both push and pull. 21.3 shows its worrking.

Lease is a contract between two entities(here server and proxy) in the distributed system. It specifies the duration of time the server agrees to notify the proxy about any updates on the web page. Updates are no

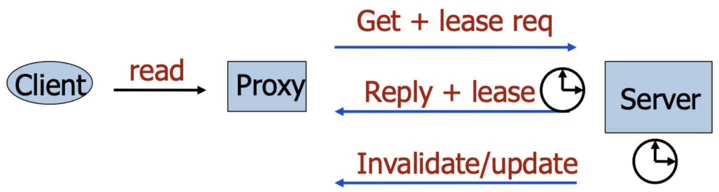

Figure 21.3: Lease

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, degenerates to polling (pull), if duration is infinite it is server push and makes the server stateful. Duration in between zero and infinite is a combination of both pull and push.

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 on 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: Does each proxy have its own lease?
Answer: Not only does each proxy have a lease, there is a lease for each webpage at a proxy. If a proxy caches 100 different webpages, each of them will have a different lease. It is not lease per proxy, it is lease per webpage 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 webpage changes for every active lease on that webpage. 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. Sever can know the age and server load. Proxies can track popularity of the webpage and report stats to the server. Monitoring framework can be added to the proxy system, it is not a part of lease approach.

21.3.5 Cooperative Caching

Collection of proxies cooperate with each other to service client requests. We will assume that there is a heirarchy as tree structure with leaf caches. Figure 21.4 explains the message flow for once single client request in the case of "Hierarchical Proxy Caching". The client requests for a web page. If it is a cache miss, the proxy will send requests to it's 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 send an HTTP request to the parent to resolve. If there are more levels in the heirarchy, the process will repeat for its parents and so on. The parent will fetch the data from the server and sends the web page as response - flows down the heirarchy to the client.

This works well when a nearby proxy actually has the content. If there is a global miss - no one in the heirarchy has the content, latency will increase significantly. As we can see, 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 server in the event of a cache miss on the proxy.

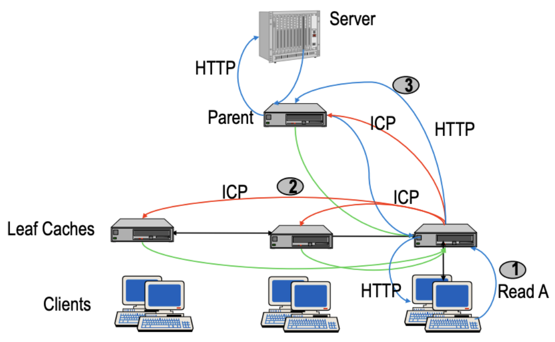

Figure 21.4: Heirarchical Proxy Caching

This is complicated as seen in the figure, as there are many arrows all over the place. To simplify, we can flatten the hierarchy. Every proxy keeps track of what is stored in its nearby caches. Keep track of what's cached using table lookup - save which node has what in the table. If the content for a request in a nearby cache, it is fetched. If there is no entry in the table for a request i.e. none of the nearby caches have the content, go to the server to get it, and add it to the table. Now there is no heirarchy, it is a flat network of proxies. Here, lookup is local. Hit is at most 2 hops and miss is also at most 2 hops. Every time you fetch or delete a page, you update the table for all the nodes. Every node keeps a global table that must be kept consistent. There is an additional overhead for keeping this consistent but performance is better.

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 orthogonal to whether lease is being used or not
Follow-up Question: If you do multicast, you don't require 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.

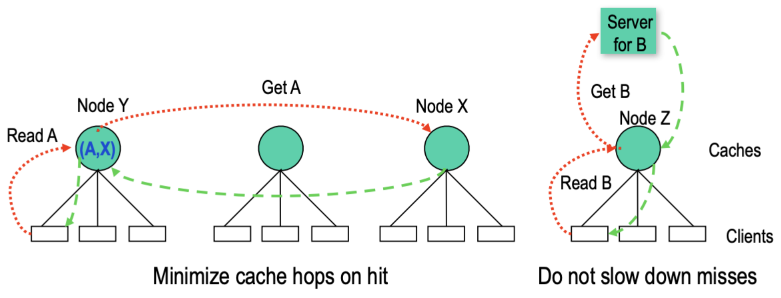

Figure 21.5: Locating and Accessing Data in the Flattened Network

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

Question: If you have heirarchical 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 far off 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. The work was 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 lots of these servers in many different networks. Servers are deployed as clusters of different sizes depending on the demand.

Content providers are customers of CDN service. They decide what content to cache. Consistency is the responsibility of the content provider.

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

21.4.2.1 CDN Request Routing

Because they have lots of different servers, when a request comes in, CDN has to decide which proxy has to serve that request. Request for the same content is sent to different proxies based on location of the user. The idea is to send the request to the server nearest to user. Request routing is routing the request to the right proxy to get it served at the lowest latency. How do CDNs do this? They have large load balancers that look at incoming requests and send it to different proxies. Typical CDNs have 2 level load balancer:

1. gloabl level - decide whcich cluster to send the request to. nearest cluster is chosen
2. local level - once request is mapped to a cluster, which server in the cluster will serve the request.

Deciding which is the closest cluster or closest server is done using Domain Name Service (DNS). DNS is a service which takes the url and gives the IP address. The browser makes a connection to this returned IP address. DNSs used by CDNs have special algorithms that change the IP address returned to the client based on the location of the client.

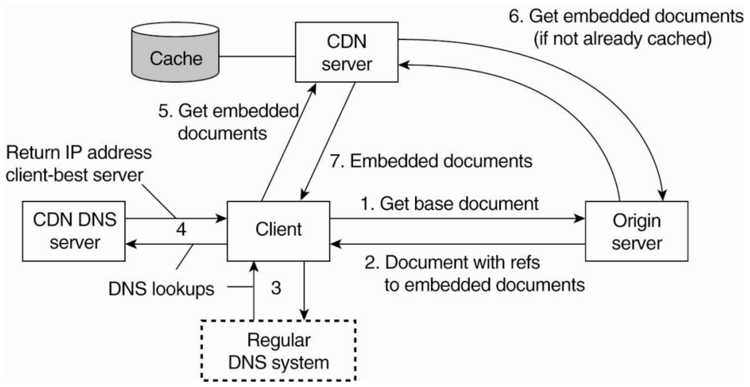
Cache
CDN server
6. Get embedded documents (if not already cached)
5. Get embedded documents
Return IP address client-best server
7. Embedded documents
CDN DNS server
4
Client
1. Get base document
Origin server
2. Document with refs to embedded documents
DNS lookups
3
Regular DNS system

Figure 21.6: CDN Request Processing

Question: Do edge proxies use cooperative caching?
Answer: In this case, no need to do cooperative caching. Essentially going to replicate 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 then 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.

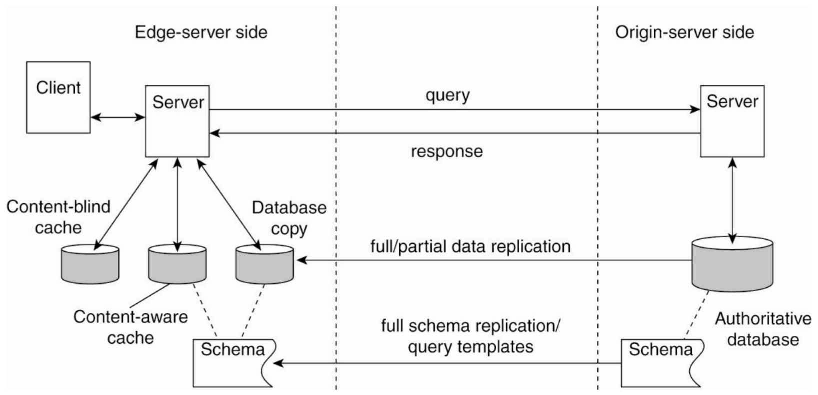

Figure 21.7: CDN Hosting web apps

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.