CMPSCI 677 Distributed Operating Systems **Spring 2019**

Lecture 19: April 17

Lecturer: Prashant Shenoy *Scribe:* ***Mikayla Timm***

In this lecture, professor wraps up the discussion of fault tolerance and starts a new topic "World Wide Web", as the topic helps in getting started with Lab 3.

19.1 Raft Consensus Protocol

Paxos is very hard to reason about. Therefore, another consensus protocol Raft is developed with an aim to make it easier to understand. It is not meant to perform everything that Paxos does, but it works in the presence of failures. In Paxos, we had to agree on operations like state of lock, order of applying queries etc. Raft is specially designed for replicated servers, where all of the servers process the same request. When a request comes in, the request is forwarded to all the replicas, so that all of them could perform the same action and be consistent. When multiple requests come in, we have to order them and send them in the same order to all the replicas.
Raft uses a technique called State Machine Replication(SMR), which became popular in distributed systems. SMR maintains a log file in each replica. This log is used to determine the order in which the requests come in. Requests are written to the logs in the same order, so the logs at all of the replicas will be the same. If we perform the operations in same order for all replicas, then they will be consistent.
Raft is not completely distributed. First a leader is elected and this leader decides an ordering and imposes the ordering on all the followers. If the majority agrees on the ordering, the ordering is committed and minority which does not agree will have to synchronize at a later time before they can execute requests.

Question: Without using Raft, how do we ensure database consistency?
Answer: Can have frontend node send the same request to all replicas, or have a master node that receives all requests and sends it to where it needs to go in a single order.

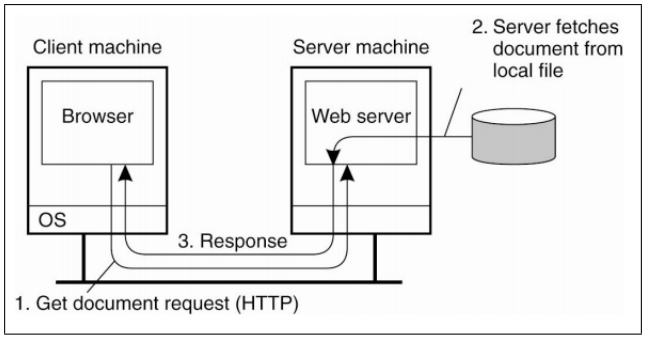

Figure 19.1: Overall Organization of a Traditional Web Site

19.2 Traditional Web Based Systems

Figure 19.1 shows a basic client-server request response protocol. Browser sends a HTTP request to the web server. The server fetches the document from database and sends back the response to the browser.

19.3 Processes - Clients

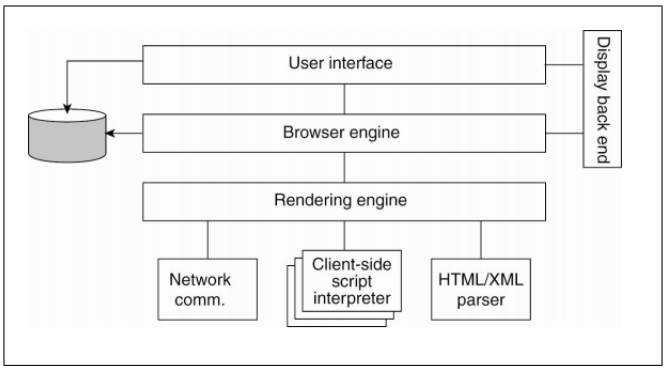

Figure 19.2: Logical Components of a Web browser

Browsers are complex with many built-in functionalities. Figure 19.2 shows the major components of a web browser. The rendering engine takes a HTML documents and renders it on the user screen. The Client side interpreter interprets the Java/PHP scripts embedded in the documents.
Proxy is an entity which lies between the client and the server. They are commonly used for web caching. Recently used web pages are cached. The browser will send requests to the proxy instead of server. Proxy will process the request. If the requested content is cached on proxy, it is will send back reply to client. Else, the proxy makes another request to the server.
Proxies can also be used for transcoding. Transcoding refers to actively changing content to better cater

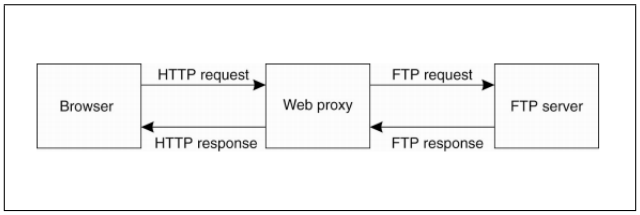

Figure 19.3: Web Proxy for Transcoding

user needs. For example, adjusting the web page layout to make it suitable for mobile display. The example shown in figure 19.3 does protocol translation. Another example is rendering video content - the proxy may translate high definition video to standard definition.

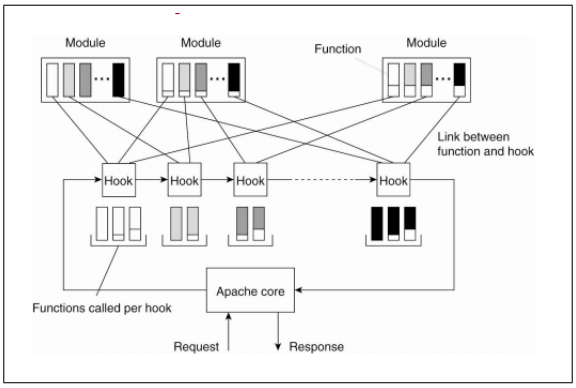

Figure 19.4: Overall Organization of Apache Web Server

19.4 Apache Web Server

Figure 19.4 is a Apache Web Server. It has a modular architecture. We need to configure a set of modules that will perform request processing. When a request comes in, it will go through a series of processing steps. Each module performs partial processing in a pipeline fashion. You can turn each module on or off.

Question: What is an example for pipeline processing?
Answer: Let's say the client has sent an HTTPS request, which means the connection is encrypted. First, one of modules decrypts the request. The next module extracts the web pages. The PHP commands in the web page are processed using an interpreter module. At the end, a module have to encrypt the response. These are some sequence of steps.
This server can also have multi-tier architecture. We can employ caching at multiple levels of these tiers. For example, database queries can be cached.

19.5 Web Server Clusters

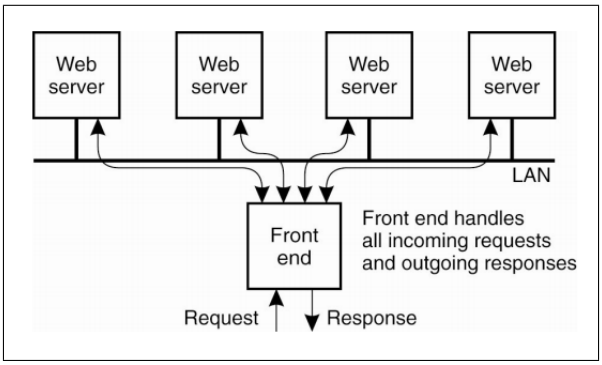

Figure 19.5: Web Server Clusters

Figure 19.5 shows an example of clustered architecture. Each box are themselves multi-tiered. Each in-

coming request can be forwarded to one of the four Web Servers. Responses flow back through the load balancing switch. The clusters can service more requests, which allows it to scale up better.

Figure 19.6 depicts another such architecture. The dispatcher is responsible for balancing the load. Each

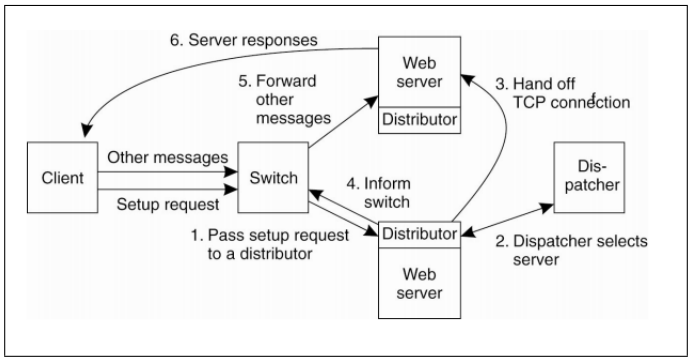

Figure 19.6: Scalable Content-Aware Cluster of Web servers

client is mapped to one of the servers. Once mapped, the switch will store this information and uses it to forward all consecutive requests from that client to the mapped server.

Question: Why is the dispatcher not dispatching the messages?
Answer: There can be many different architectures. The specific example shown here is doing session level dispatching. This means that every user is mapped once to a web server. From that point, the requests from that browser will be sent to the same server without further need of dispatching.

Question: Can the switch be a bottleneck?
Answer: If we have high volume of requests, the switch could become a bottleneck. It is simply forwarding the request. We can make it distributed as well. If we are not doing much processing and if implemented in hardware, it can support high request rate.

Question: When the session begins, do you talk to the dispatcher?
Answer: Yes. We typically go from a web server to the dispatcher. Dispatcher selects the server. It informs to the switch. And from that point switch takes over.

19.6 HTTP Connections

Figure 19.7 shows the original HTTP 1.0, where browser sets up a new TCP connection to the server every time we make a HTTP request. The connection is then set down. If there are hyper links, it will create new connection. Making new connection to the same server every time is wasteful. A variant, version 1.1 Persistent HTTP from figure 19.8 uses a single connection and piggy backs requests. The browsers does not close the connection immediately, in anticipation of further requests. This reduces the overhead to the server, as it limits the number of socket connections.

Question: This has to enabled on the server side?
Answer: Yes. If the server can support HTTP 1.1 then we can use it. The first time when browser sends a request to the server, it asks for the version of HTTP.

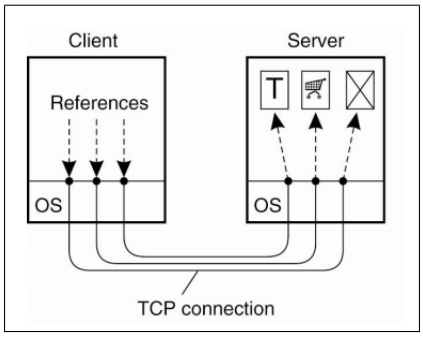

Figure 19.7: HTTP 1.0: Using Non-Persistent Connections.

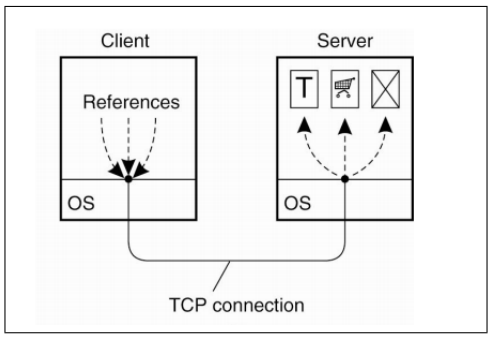

Figure 19.8: HTTP 1.1: Using Persistent Connections.

Question: How does the client browser differentiate between the two types of dispatcher?
Answer: The client browser should not even know about the presence of dispatcher.
One of the design issues for HTTP 1.1 is the duration of time to keep the connection open. Should the browser wait for 5min or 1hr? This is a design decision. As server will receive requests from multiple other users, this problem is even worse because of open socket connections. Usually, server closes the connection, if the connection is ideal for long time.

19.7 HTTP Methods

HTTP protocol has five commands:

1. The simplest among them is the 'GET' command. This command allows the user to request and get a document.
2. While 'PUT' command is used to send the document to the server. 'PUT' is only used if the server have WebDev service.
3. 'POST' is the most common command. It allows the user to send information back to the server.
4. 'DELETE' can be used to delete a page on server.
5. 'HEAD' gets the header of the document. It is typically used for caching.

Question: Which command can be used to update information on the server?
Answer: First, do a 'GET' to get the web page with form embedded in it, once you fill the form and submit it will go as a 'POST' request.

19.8 Web Services Fundamentals

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
      <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
        <n:priority>1</n:priority>
        <n:expires>2001-06-22T14:00:00-05:00</n:expires>
      </n:alertcontrol>
    </env:Header>
    <env:Body>
      <m:alert xmlns:m="http://example.org/alert">
        <m:msg>Pick up Mary at school at 2pm</m:msg>
      </m:alert>
    </env:Body>
</env:Envelope>
```

Figure 19.9: Example of XML-based SOAP message

HTTP is not only confined to typical interactions between users through browsers. It also facilitates process to process communication. In this case, the two applications communicate without human interaction. RPC is an example of such communication. SOAP (Simple Object Access Protocol) was initial protocol used to make RPC request over HTTP. The entire RPC request is sent as a XML document. The server after receiving the document, parses it, perform required operation and sends back the response as another XML document. Figure 19.9 shows one such XML request document. Here the client is calling 'alert' method and passing the string 'Pick up Mary at school at 2pm' as an argument to that method.

19.9 Restful Web Services

As we can see, for calling single method with one argument, we have to send such a long XML file. SOAP did not perform well because of this overhead. As a result, SOAP evolved into Restful architecture. Restful architecture also makes RPC request over HTTP. In case of Restful architectures, the communication is light weight and assumes one-to-one communication between client and server.
Question: What do you mean by saying 'SOAP requires arbitrary servers'?
Answer: SOAP is designed for an arbitrary distributed application, where there are arbitrary number of nodes, that are all communicating over RPC to implement a larger application. Each edge connecting the nodes in figure 19.10 (i) is a SOAP interface. So, SOAP is designed for general distributed application scenarios. On the other hand, Restful web services(19.10 (ii)) assume one client is talking to only one server through HTTP based REST requests. We assume there is no state. It will simplify a lot of things.

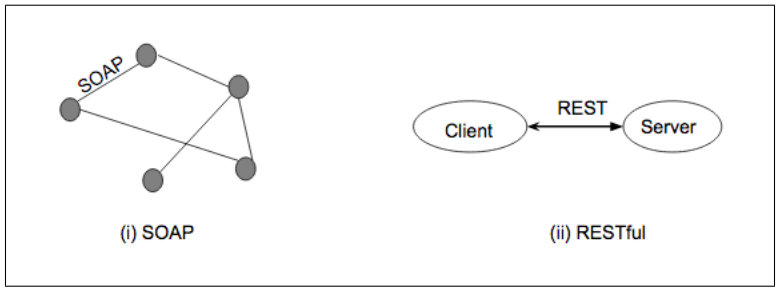

Figure 19.10: SOAP vs RESTful Connections

In the case of RPC requests, Restful services will send the method name and arguments in the URL itself, as opposed to sending a complex XML document as we did in the case of SOAP.
Question: How do you send string where there is a space in them in the request URL?
Answer: You will use control characters to describe it. For example, "IBM Research" can be encoded as "IBM%20Research". It depends on the HTTP specification.
Question: Is there any limit on the length of the characters?
Answer: There should not be any such constraints.

Question: Why is the XML document a problem in the case of SOAP?
Answer: Parsing the XML document is a processing overhead. It effects efficiency. In the case of Restful architecture processing overhead is minimized as the method name and parameters are part of URL.

19.10 SOAP VS RESTful WS

1. SOAP application can be written in various languages and can run on different platforms. It need not use HTTP. Whereas, RESTful services only support HTTP.
2. SOAP supports arbitrary set of inter-communicating applications. Where as REST uses point to point connection with no intermediaries.
3. SOAP has a wide set of standard, whereas RESTful services does not have any pre-defined standards
4. Over head is higher with SOAP compared to REST.
5. Rest has less of a learning curve compared to SOAP.

19.11 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 19.11 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, it will reach out to the near by proxies to get the data. This can make fetching faster. The cache becomes the union of all caches.

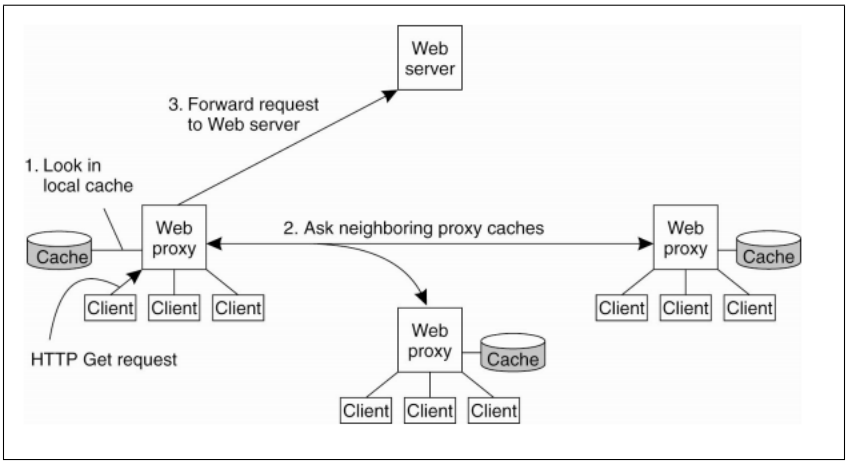

Figure 19.11: Cooperative Caching

19.12 Web Caching

It is also import to deal with consistency. Web pages tend to change with time. 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. The popularity and update frequency can be different across web pages. We need to consider both these issues for maintaining consistency. We need to make sure the caches have the most new/updated version. There are different methods for this:

1. Client Pull VS Server Push
2. Invalidate VS update:

   Invalidate: server just notifies the proxy that web page has changed. The proxy will then remove the page entry from it's cache.

   Update: server actually sends the updated page to the proxy.

19.12.1 Push-based Approach

In this approach, the web server will keep track of web pages requested by every proxy so that it knows which proxies cached each page. Whenever a web page is changed, the server will either a notification(invalidate) or the new version(update) to the set of proxies which cached this page. If the web page is popular it makes more sense to send updated version to the proxy. Else, if it is less popular, it makes more sense to send an invalidate. Push-based approach provides tight consistency guarantees. Proxies can also be passive, as the server does all the work in this approach. 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.

Question: What happens in the case of invalidate?
Answer: Once the proxy receives invalidate message from the server, it will delete the entry from the cache. Later, when some client requests for that web page, the proxy gets a cache miss. So, the proxy will fetch it from the server.

Question: Is caching with proxies similar to edge servers?
Answer: Proxies can be deployed anywhere in a network. Edge servers put computation specifically at the edge of the network.

19.12.2 Pull-based Approach

Proxy is responsible to maintain consistency. It polls the server periodically to check if the previously cache 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. For web pages which changes very rarely, frequent pooling is extremely wasteful. There is no such wasteful messages in push based approach. If the page changes much more frequently than polling frequency then the proxy might cached outdated content for longer times. Polling frequency should be decided based on the update frequency of web page. There are two ways to do this:

1. Server can assign a 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.
2. Proxy has intelligence to dynamically figure out the polling times. Poll duration is varied based on the observed web page updates.

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.

19.13 A Hybrid Approach: Leases

Lease is a contract between two entities 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 longer sent by the server after lease expiration and the server will delete the state. Proxy can renew the lease.

19.13.1 Policies for Leases Duration

Lease duration is an important parameter. Zero duration is a pure proxy pull approach which result in increased polling requests. Where as, infinite lease is server-push approach which makes the server stateful. 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 less lease duration. This will remove the burden of storing proxy state on server side.

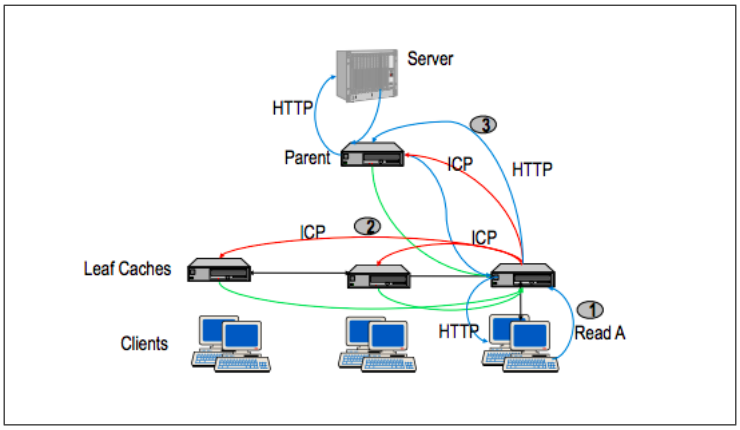

Figure 19.12: Hierarchical Proxy Caching

Question: In the internet, who owns the proxies?
Answer: It depends on the scenario. We can get open source proxy and the user can point the proxy to the browser. Mostly, proxies are run by content distribution networks. They deploy proxy as a service to the end users.

Question: If the proxy request lease of certain duration, should the server provide it?
Answer: There are two options. One option is proxy can request for lease, but server decides the duration. Other option is proxy requesting the duration. First option is better.

19.14 Cooperative Caching

Figure 19.12 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 contact it's peers(red arrows). If none of them 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. The parent will fetch the data from the server and sends the web page as response.
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 effect 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.

This is complicated as seen in the figure, as there are many arrows all over the place. To simplify, we can flatten the hierarchy. Keep track of what's cached using table lookup - save which node has what in the table. If there is no entry in the table for a request, go to the server to get it, and add it to the table. 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. All lookups are local.

19.15 Content Distribution Network

Companies set up large numbers of caches throughout the world that hold all kinds of content. The DNS figures out which proxy is closest to you.