

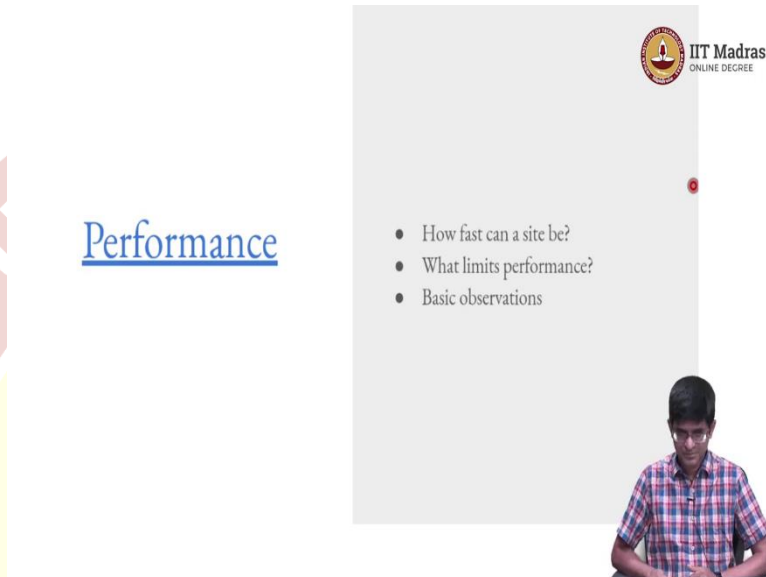
IIT Madras

ONLINE DEGREE

Modern Application Development – I
Professor: Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Performance of a Website

Hello, everyone, and welcome to this course on modern application development.

(Refer Slide Time: 00:16)



So, now, with all of this in mind, now we know how a web server works there are many issues that come up in the context of performance. And what we mean by that is, how fast can a site operate? How quickly can it respond to requests? What are the things that limit the performance? So, let us try and look at some basic observations. These are just sort of back of the envelope calculations, so to say, that I am going to show over here, but they are important to keep in mind when you are developing an app.

Probably not in the beginning stages, when you are just sort of starting out with an app, maybe you do not need to worry too much about these details, but at some point, hopefully, whatever app you are creating is something that you want to scale and you want more people to use it. At that point, there are a number of issues that come into the picture that you need to keep in mind. It is worth sort of at least understanding those in the context of the server right now.

(Refer Slide Time: 01:11)

Latency



- Speed of light: 3×10^8 m/s in vacuum, $\sim 2 \times 10^8$ m/s on cable
 - ~ 5 ns / m $\Rightarrow \sim 5$ ms for 1000km
- Data center 2000km away
 - One way - request - ~ 10 ms
 - Round-trip 20ms
- Max 50 requests / second



So, one of them is what is called latency, and latency essentially refers to how much time does it take to get the response for a given request? I ask for something, the server does some work, and finally comes back with the request. What is the time between the time that I sent the request and the time that I got the response? How much time did that take?

Now, we are used to thinking of computers as being very fast, and we also think of the speed of light as being extremely fast, but surprisingly enough, the speed of light can actually limit the kind of performance that we can get on servers. So, as we know, the speed of light in vacuum is about 3×10^8 meters per second, I have written it as $3e8$, the scientific notation.

Now, on a cable, it is a bit less than that. So, on copper cable it is probably around 2×10^8 I think on optical fiber also it is somewhere around 2×10^8 meters per second. So, that is extremely fast. For most purposes we cannot think of anything even remotely approaching the speed of light.

But what does this actually translate into? It means that for light to cover one meter it takes about 5 nanoseconds 2×10^8 meters per second can also be interpreted as 5 nanoseconds to cover 1 meter, which means for a distance of 1,000 kilometers, 10^6 meters is going to take me about 5 milliseconds.

So, what does five milliseconds mean? This is for 1,000 kilometers. Let us say I have a data center 2,000 kilometers away. I am in Chennai and I am connecting to a data center in Delhi,

roughly 2,000 kilometers ballpark. One way, the request is going to take 10 milliseconds, which means that the round trip my request goes from my computer here to the server in Delhi 10 milliseconds response, even if the server is instantaneous the response takes another 10 milliseconds to come back. So, I have a round trip of 20 milliseconds.

What does that mean? It means that if I am continuously sending requests to the server, and I want it such that every time I send a request, I want to get back the response before I can then proceed, I am limited to a maximum of 50 requests per second. So, even the speed of light, which is something which we otherwise think of as unimaginably fast can be a limiting factor even on the earth itself.

Forget about if you are going to satellite communications. Once I go through a geostationary satellite link, the time is significantly larger, that is 30,000 kilometers or so each way. So, if I have things of that sort, I might find that just the latency itself can start to become an issue 50 requests per second does not sound like a whole lot. And 20 millisecond latency is small, but it is not something you casually want to deal with all the time either.

(Refer Slide Time: 04:41)

The slide is titled "Response size" and features the IIT Madras logo in the top right corner. It contains the following bullet points:

- Response = 1KB of text (headers, HTML, CSS, JS)
- Network connection = 100 Mbps
 - ~ 10 MBytes/s
- ~ 10,000 requests / second

In the bottom right corner of the slide, there is a small video inset showing a man in a checkered shirt speaking.

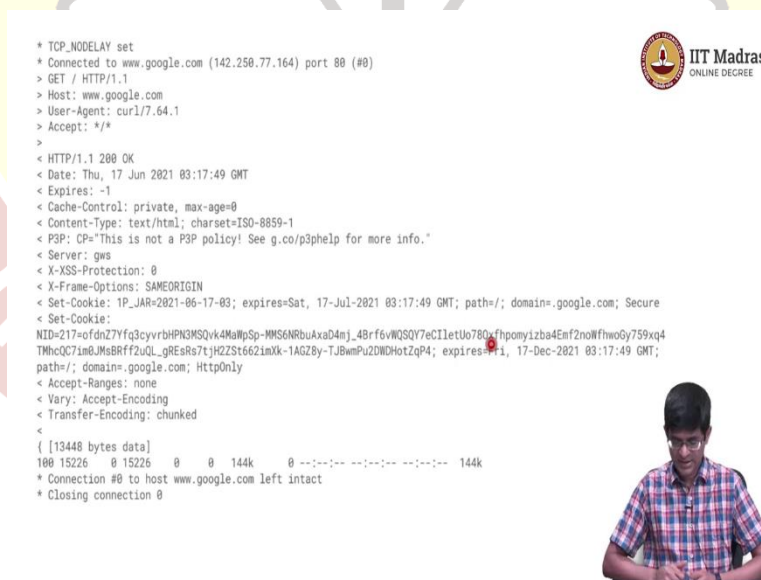
What about a server that needs to send back responses? Now, we already saw what the kind of responses that a server is going to give for a given request. Even just a connection with curl, it has to give back the HTTP headers. It has to basically say the status code, it has to say the content type, the content length, the last modified date. All of that information is typically a set of headers, then it has to add on the HTML file.

Along with HTML, it might have something some JavaScript, which needs to be executed or it might have some CSS, which is used for the styling. All of those together, let us say that we have about 1 kilobyte of text that needs to be sent back for each request, and if I have a network connection, a fairly fast network connection of let us say 100 megabits per second or so 100 megabits per second is roughly 10 megabytes per second, it is probably 100 by 8, but there are typically some protocol headers, so 10 megabytes per second is probably a good estimate of what you can get in practice.

So, 1 kilobyte per response, 10 megabytes per second that limits you to 10,000 requests per second. In other words, that entire pipe that I have that 100 megabits per second pipe that I have is fundamentally not capable of handling more than about 10,000 requests per second.

Now, just imagine, even in India, let us say something like the, there is some entrance exam results being declared and if it is all trying to come out from one server and the server has a 100 megabit per second connection to the Internet, just imagine what happens when everybody tries to connect at the same time? 10,000 requests per second, server crashes. It cannot, the network connection is gone. It basically cannot handle that kind of connectivity anymore.

(Refer Slide Time: 06:38)



Now, this big chunk of text over here, you do not need to worry about it. What matters is, I basically did a get request to google.com, just using the same curl that we have already seen. And this is the response that I got. So, you can see that, first of all, it has a whole lot more information in the responses. The first line, of course, has to be the HTTP status code, which as we can see is the 200, okay, good news.

It also gives the date it says something about when this information expires, cache control, content type lots of stuff, which even I am not entirely sure of what exactly it is. The important part is what finally comes over here, which finally says that 144 kilobytes of data were transferred.

Now, you are all familiar with what the Google web page, homepage looks like, it is just one small text box over there, but if you do a request, you will realize that it is actually transferring about a 100 plus kilobytes of information just to display that small piece of the text box. Why is that? Well, Google has a lot of other things that are going on there.

They have some tracking information, they have JavaScript, they have things which can dynamically modify that page based on what you have been doing previously, and so on or whether you are logged in or not, lots of other information comes into the picture. The bottom line is it is around, like 100 kilobytes of text for one request.

(Refer Slide Time: 08:14)

The slide features a large, faint watermark of the Indian Institute of Technology Madras logo in the background. The slide content includes:

- Google response
- Headers - ~ 100B
- Content: 144kB
- Approx 60,000 requests per second (maybe)
- ~ 80 Gbps bandwidth

In the bottom right corner, there is a small video inset showing a man in a plaid shirt speaking.

What does that mean? It means that the content which comes from a Google response is approximately 100 plus kilobytes per second. Now, searching around on Google naturally, where else can I search shows that, at least within the last few years, at some point, there was information which indicated that Google, Google search, the one that most of us use requests, receives approximately 60,000 requests per second, not per day, per second.

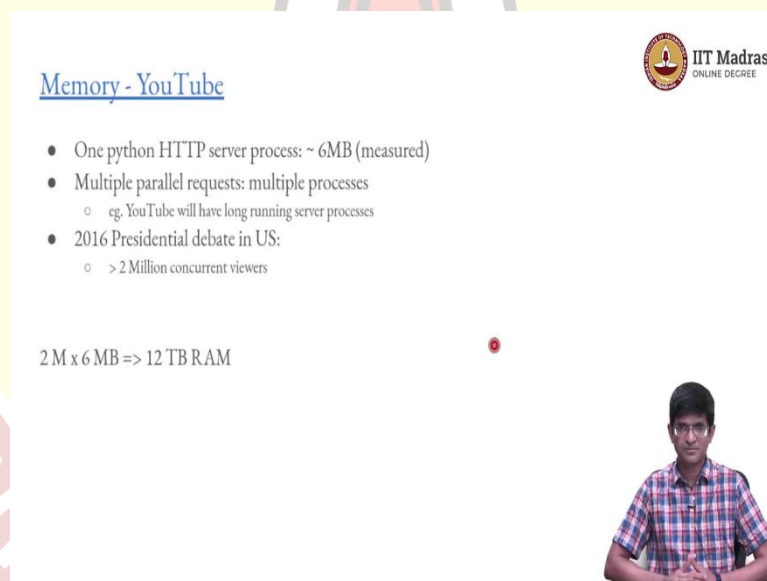
Now, it might probably be a bit more than that, because this number is only going to keep increasing. So, what does 60,000 requests per second translate into, in terms of the response that it has to give? If each of those 60,000 requests per second requires these 100 kilobytes of

information to be sent back, it may not. I mean many of those requests may not require all of that data, but worst-case scenario, this is roughly what it is.

And in fact, many of the requests will actually have a query, which means that they will need even more information coming back. So, this may be a reasonable average. The point is, I now have 60,000 requests per second for each of those requests, I need to send back some 140 kilobytes of data.

That means that Google servers effectively need an aggregate bandwidth of something like 80 gigabits per second, clearly not something that a single server can handle. Probably not even a single data center. There probably is not a data center that will have that kind of bandwidth to the outside world. This is why they need to be distributed across centers all over the world. And to reach this kind scaling where you are able to manage 60,000 requests per second, you really need to be able to scale out in different ways.

(Refer Slide Time: 10:10)



The slide is titled "Memory - YouTube" and features the IIT Madras Online Degree logo in the top right corner. It contains a bulleted list of memory usage examples:

- One python HTTP server process: ~ 6MB (measured)
- Multiple parallel requests: multiple processes
 - eg. YouTube will have long running server processes
- 2016 Presidential debate in US:
 - > 2 Million concurrent viewers

Below the list, it states: $2\text{M} \times 6\text{MB} \Rightarrow 12\text{TB RAM}$. In the bottom right corner, there is a small video inset showing a man in a plaid shirt speaking.

A similar computation for YouTube. So, the Python HTTP server that I was running, I ran a command called top, which sort of tells me how much memory is being consumed by each of these processes. And that told me that at least on my Linux server, that Python, that single Python process was consuming somewhere around 6 megabytes or so. It was using up that much of the ram of my system.

Now, 6 megabytes is not a whole lot, because my machine has something like 8 gigabytes of RAM, so 6 megabytes is nothing. But let us start looking at parallel connections. Every time I have a new connection, I need to spin off one new process to respond. Now, ideally, that

process would be there, it would respond and then go away, it would free it up for handling the new connection. But what if I had long running connections? Something like YouTube where people are connecting and then stay connected for a very long time.

If each one of them was spinning up something like 6 megabytes of memory, think about what happens for something like the 2016 presidential debate in the U.S that was one of the most watched events in YouTube history, at least at that point. Now, there might have been more, but this is already a big enough number 2 million concurrent viewers.

So, effectively, these are people who are viewing a video stream, they are going to be connected for a significant amount of time, and 2 million viewers, if each one of them required a server that occupies something like 6 megabytes of RAM, we are talking about 12 terabytes in total. Big servers today would typically have like 30, 60 gigabytes. I mean, those are reasonably servers, big servers you could probably have 512 gigabytes are there.

1, 2 terabytes, maybe. 12 terabytes on a single machine is I have not heard of such a machine. They might exist, because I have actually seen machines with two terabytes of RAM. There is nothing fundamentally preventing a machine from having 12 terabytes, but that is not the way you would do it. You would not try and build one machine with 12 terabytes of RAM to handle this. What you would do instead is try and spread it out across thousands of servers.

So, if I had 1,000, servers, things look better. Each of those servers needs only 12 GB of RAM. The point that I am trying to make over here is that all of these are issues that you need to keep in mind when you are deciding how your app is going to get actually used at the end. Are these things that concern you right at the beginning? Should they be things that concern you after your app has become popular?

Should you be thinking about how to design it right from the beginning? In other words, especially things like does Google really need to send us 100 kilobytes of information for each request to the main web server or could it have just sent us a few 100 bytes? And then depending on what we want to do, it scales up the amount of information that it sends. Could there be a better use of the network bandwidth? Could there be a better way that I could size my servers in order to respond?

(Refer Slide Time: 13:27)

Storage - Google



- Index 100s of billions of web pages (funny cat videos?)
- Cross-reference, pagerank
- Total index size estimate:
 - 100,000,000 Gigabytes => 100 Petabytes
- Storage?
- Retrieval



All of these are issues that we need to keep in mind. Another thing like I said, we looked at network, we looked at memory. Storage is another thing, so Google, for example, indexes hundreds of billions of web pages. They have to be cross referenced page ranked. I mean, how many links are going into a page determines how popular or how important the page is.

An estimate, which once again comes by googling. When you just go search on Google for what is the size of the Google database and it comes up with a number which is from one of their own internal blogs, which estimates that it is somewhere in the range of 100 petabytes. Now, we have mega, giga, tera and then we have peta. So, 100 petabytes is a 100 thousands terabytes or a 100 million gigabytes, a pretty large number.

So, first of all, how do you even store that kind of information? You cannot be doing it on disks, which can store like, even if a disk stores 1 terabyte, you are talking about 1 lakh of those discs to store all of this information. But clearly it has to be done that may be some way by which you can actually have one lakh discs or even more than that, and store it in a way that you can actually retrieve the information that you need when you need it.

(Refer Slide Time: 14:50)

Summary

- The Web is a useful device/OS agnostic platform for apps
- Built on HTTP for transport, HTML and related tech for presentation
- Servers trivial at most basic
- Scaling requires careful design



So, to summarize all of this, what is the point of all of this exercise? The web by itself is a very useful device. It is operating system agnostic and it is a good platform that we can target for building apps. It is built on solid foundations HTTP, as the protocol for transport. And as we will see later HTML and related technologies which are used for presentation or how you view the output.

The most, at its most basic level, the concept of a web server is a very trivial thing. All that it does is basically receive requests and send back information. And writing a web server is not a complicated task. Writing a useful web server something that will do the kinds of things that you want that will scale the way that you want is a tricky task.

And especially scaling, when you start seeing that the number of requests that you are getting or the number of clients or the number, the amount of storage, the data that you need, as that starts increasing, it may not be possible to just keep on adding more resources. You may actually have to go back and redesign your protocols and your computation itself in some way so that you can scale accordingly.