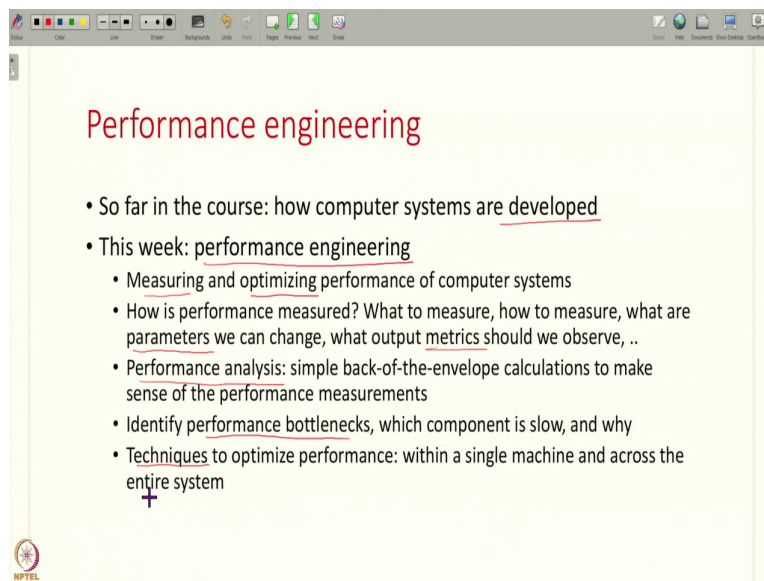


Design and Engineering of Computer Systems
Professor Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 45
Performance measurement

Hello all welcome to the thirty first lecture in the course design and engineering of computer systems. So, in this week we are going to start a new topic, which is performance engineering. So, let us get started.

(Refer Slide Time: 0:30)



Performance engineering

- So far in the course: how computer systems are developed
- This week: performance engineering
 - Measuring and optimizing performance of computer systems
 - How is performance measured? What to measure, how to measure, what are parameters we can change, what output metrics should we observe, ..
 - Performance analysis: simple back-of-the-envelope calculations to make sense of the performance measurements
 - Identify performance bottlenecks, which component is slow, and why
 - Techniques to optimize performance: within a single machine and across the entire system

+

NPTEL

So far in this course, we have seen how to build, how to develop, how to design computer systems. In this week, we are going to study how to engineer the performance of these computer systems. So, right at the beginning of the course, we have seen that, of course, a system has to satisfy certain functional requirements, it has to do the job that it is built for. In addition to that, it also has to give good performance. This is an example of a non-functional requirement.

So, in this week, the topics that we are going to study related to performance are as follows. So, first, we will study how to measure performance, given a computer system, what does it mean to measure performance? What will you measure? How will you measure? What parameters will you change? What are the output metrics that you will observe? All of these things, the basics of what is performance, what how to measure performance, we will study. And then the next thing we will do is we will see how to do a simple performance analysis fine, you have measured performance, you got some numbers, but to the results make sense.

How do you estimate, what is the right value? Are you close to that right value? So, some simple performance analysis how to do we will study. And then the next of course, the next big chunk is how to optimize performance of a computer system, identify what are the performance bottlenecks, why is the performance poor if it is poor, which component is responsible for it and how to optimize that component?

And of course, we will see various techniques to optimize performance both within a single machine as well as across the entire system. So, the one thing I would like to point out is this performance engineering by itself is a full course, it is a full-fledged course. So, in this one week, what I am going to do is describe only the high level concepts. And if you are interested in this, you should go ahead and take a complete course to understand this fully.

So, now, let us start with an example, when we talk about performance, instead of talking about it at a high level, let us take an example application.

(Refer Slide Time: 2:38)

Example: multi-tier web application

- Consider a multi-tier application (many examples seen so far)
 - Front-end web servers, multiple application servers, backend databases
 - Clients make requests, processed by system, responses sent back
- Incoming traffic into the system can be measured using: *load?*
 - Number of concurrent clients connected to the system
 - Incoming rate of requests per second
 - Mix of various types of requests in the incoming traffic
- Performance of the system measured by (average values of):
 - Throughput: number of requests per second handled successfully
 - Response time / RTT / latency / delay: time taken by the system to return back a response to a client request (varies by type of request)

So we have been seeing this, multi-tier application for a while now. And we have seen many examples of this last week also. So, a typical multi-tier application, how does it look like there will be multiple, say, front end servers, web servers that are getting traffic from a whole bunch of clients, then these front end web servers will talk to various application servers to handle various types of requests. And these application servers may contact various databases, and in the end all of this happens and some responses sent back to the client. So, this is a simple model, let us keep this in mind.

Now, to this system, how do you measure the incoming traffic? What is the load, if you want to say the system is under a lot of load, the system is facing a lot of load, what does that mean? What do we mean by the load? So, this load or incoming traffic can be measured in multiple ways. For example, you can count the number of concurrent clients if this is an E-commerce website, you can say that a million clients are connected to this website at this point, and they are trying to do online transactions that is one way of measuring.

The other way of measuring the load or incoming traffic into the system is the rate of requests e.g. this system is getting a million requests per second or 10,000 requests per second these requests can be anything of course, there are various types of requests. And this mix of requests can also be specified. For example, it is getting 10,000 requests per second to search products and 20,000 requests per second to buy products, something like that, there are various types of requests and the mix of that can also be described.

So, all of these things describe the load or the incoming traffic into the system. Then what is the performance of the system given that all of this traffic is coming in? How do you measure the performance of the system? There are roughly two main metrics. One is called throughput that is the amount of work done per second number of requests per second that have been handled successfully.

So, suppose your system is able to process 10,000 search requests per second or it is able to process 20,000 online purchases per second. So, much of traffic is coming in and the system is able to handle it. So, that is called the throughput of the system and the other measure is the what is called the response time it is also called by different ways like RTT, latency, delay. RTT stands for round trip time. There are many names which are more or less the same thing, what it measures is the time.

The time taken by the system so, if I click on a button saying buy or if I click on a button saying search for a product, how much time does it take for the system to go through all the components do the processing and return a response back to me that time is what is called response time. Note that these are two different metrics, the system can have very high throughput and but also take a long time to respond it can have very low throughput, but can be quick to respond any combination of these two are possible.

So, in general, every system we measure it by throughput, how much of work is it doing per second and how fast is it doing that work when a request comes in how fast is it returning a

response to us? So, now, why does the system have a performance issue, any system if you look at it, there is always a performance bottleneck. So, let us understand what we mean by a performance bottleneck. So, all of you intuitively know what is a performance bottleneck.

(Refer Slide Time: 6:41)

Performance bottleneck

Diagram illustrating a web application architecture with three components: Front end (1000 req/s), app server (5000 req/s), and DB (100 req/s). The DB component is circled and labeled as the bottleneck.

- The performance of a system is limited by the slowest component in the system (bottleneck)
- Consider a web application servicing one type of requests
 - Front-end can serve 1000 req/s, app server can handle 5000 req/s, backend database can process 100 req/s
 - Max throughput of the system is 100 req/s (capacity)
 - Database component will be the performance bottleneck and will be fully occupied at peak input load (other components will not be as busy)
 - Database component takes approx. $1/100$ seconds = 10 milliseconds to process each request (service demand)
 - Response time of system will be at least 10 millisecond + time needed at other components
- Sometimes, the network can also be a bottleneck, not any component
 - Some switch on path between clients and server can only forward 50 req/s

If you have many components in a system the slowest component is usually called the bottleneck it limits the performance of the system. For example, if you have many roads, you have a very wide road and then you have a very narrow road through which very little traffic goes and then another wide road then all the traffic will get jammed here at the entry to this narrow road.

So, this narrow road is your performance bottleneck and the traffic on this entire highway can only flow as fast as this narrow road. If this narrow road is sending 10 cars per second only that much traffic can also flow on the other roads also because 20 cars even if they come here, they will stop over here. So, the performance of a system is limited by the slowest component and that slowest component is called the bottleneck.

So, if you take our example of a web application, you have a front end and you have multiple application servers and then you have a database and the users request goes to the front end, the application server and the database and then a response is sent back. So, suppose this front end can process 1000 requests per second, this can process 5000 requests per second and this database can process 100 requests per second.

So, what will be the maximum throughput of the system if a lot of load is coming in the system will in the end be slowed down by the slowest component it can only do 100 requests per second. So, that is called the capacity of the system. So, the maximum throughput that the system can give is called its capacity and this is limited by the slowest component in the system.

And when the system is handling a load of the maximum load, it can handle 100 requests per second what will happen this whatever is the slowest component, this database component will be fully occupied 100 requests per second are coming in. The front end application server are not that busy, because they can do a lot more work, they are only doing 100 requests per second, but this database will be fully occupied. So, that is why it is the performance bottleneck.

And any more load of comes into the system, the system will be overloaded because this database cannot do any more work than that. And the other terminology that I would like to introduce is what is called service demand. That is how long does each request approximately take to serve. So, why is this database only able to handle 100 requests per second, because most probably each request is taking 100 of a second that is each request is taking 10 milliseconds to process most likely.

So, note that I am using very approximate terminology here in if you study this formally, with all the math and everything you will use distribution, each request may not take exactly fixed amount of time, there will be a distribution around this average. But I am taking a very simplistic view in these next few lectures to make you understand these concepts intuitively and not going into a lot of rigorous math. So, please keep that in mind. So, if a database is handling 100 requests per second, most likely each request is taking one hundredth of a second.

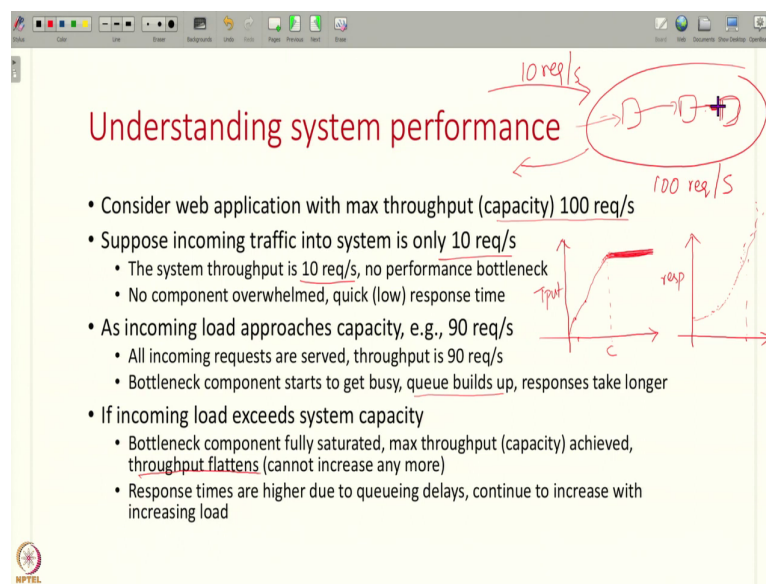
That is why you are able to do 100 of these in a second and not more not less. So, this time taken by each request to be processed this time to service each request is called the service demand of this component. And usually the capacity how much a component can process will be the inverse of this service demand. If each request is taking 10 milliseconds you can do 100 requests per second and the response time of your system will be at least this 10 millisecond you will at least need 10 millisecond here plus whatever time is needed at the application server front end all of this if you add that will be the response time of your system.

So, this is what I would like you to visualize when we say that a system has a performance bottleneck what does it mean there are there is a pipeline of many components the slowest component has a certain maximum throughput and that becomes your maximum throughput that the entire system can give you and that becomes your performance bottleneck that becomes the capacity of your system. And this capacity is dictated by the service demand, how long does that bottleneck component taking to process a particular request?

And all of this is of course, assuming this system is the bottleneck sometimes it can so, happen that between the client and the server some networks which can also become the bottleneck, some switch on this path is only forwarding 50 requests per second. Then the question of the database becoming the performance bottleneck never arises the database never even sees more than 50 requests per second then of course, it is not the bottleneck. So, your bottleneck can be elsewhere also.

So, usually the network or the system or at the client itself, may anywhere there can be a performance bottleneck in the system.

(Refer Slide Time: 11:43)



So, now, let us see how do systems perform? Now that we have understood what is the basics of performance and bottleneck. Let us understand if we measure the performance of the system, what kind of values will we get and how do we go about measuring performance? So, again, consider the same example of a web application which has a capacity it can only handle a maximum of 100 requests per second. So, now, there is this system that has many components, and all of this together can handle 100 requests per second.

Now, suppose you have 10 requests per second coming into the system, only 10 requests per second, then what will be the throughput of the system how many requests per second can the system handle? It will handle 10 requests per second. Of course, it can do up to 100 but if you are only giving it 10 requests, the throughput will also be the 10 requests will be handled and sent back. So, the throughput will be equal to the incoming traffic as long as this traffic is somewhat low.

And of course, the response time also if you measure it will be fairly low, because no component is overloaded, everybody is comfortably doing the job the response time will be low. Now, as this load keeps on increasing, so, let us try and visualize this in a graph as your load keeps on increasing. So, initially, your throughput if your load is low, the if your incoming load is 10, your throughput is 10, if your load is 20, throughput is 20 and so on the throughput will be increasing as you increase the load.

But once you reach close to your system capacity, still your throughput will be equal to load, but what happens to the response time? Initially the response time is low there is very little work and then all the components are returning responses fast, but as you keep on increasing the load what happens now this bottleneck component is seeing 90 requests per second, 95 requests per second then what is happening slight congestion starts to build up. These requests sometimes they come in a bunch and when the requests come the database is already busy due to the previous work.

So, they end up getting queued up here, the congestion starts to build up. Because of that the queuing delay, the queues build up, queuing delay increases and your response time slowly starts to increase. So, this is your throughput and this is your response time. So, as you reach capacity, throughput is still increasing, but your response time is also increasing and once your load exceeds capacity, so this is your capacity of the system. Once your load exceeds capacity suppose you are sending 1000 requests into the system, what will be your throughput be?

Your throughput will kind of flatten out, your throughput will not increase throughput was initially increasing 10, the throughput was 10, 20 requests came in throughput was 20 request, 90 requests came in throughput was 90 requests. But if 200 requests come in your throughput will not be 200 requests because your system can only do a maximum of 100 requests per second. So, this is called throughput flattens out. The maximum throughput is reached it flattens out.

And the response times of course continue to increase. As you get closer and closer to capacity as your system becomes more and more overloaded your response times will keep on increasing because a queue is just piling up in front of this bottleneck component. This is roughly what you will see when you increase load on your system. This is the performance you observe.

(Refer Slide Time: 15:16)

The slide is titled "Understanding overload" in red text. It contains a list of bullet points describing the effects of system overload. To the right of the text, there is a small diagram showing a queue of requests (represented by rectangles) waiting to enter a server (represented by a circle). An arrow points from the queue to the server. Below the server, there is a small box labeled "Error" with an arrow pointing to it, indicating that requests may result in errors due to overload.

- What happens when incoming load to server greatly exceeds capacity?
 - Throughput flattens, cannot increase beyond bottleneck capacity
 - Response times keep increasing as requests get queued up at bottleneck (high queueing delay)
 - App software cannot process requests, returns error messages (e.g., HTTP server returns code of "503 service unavailable" if it is overloaded)
 - TCP sender does not get acks, terminates connection (socket syscalls fail)
 - Networking routers/switches/NIC may get overloaded, packets fill up queues inside router or device driver, packets are dropped
- Result of overload: very high response time (for requests that complete) or errors (requests get no response at all, or get error messages)

So, now let us understand this overload a little bit more suppose a lot of traffic is coming into the system that greatly exceeds its capacity or capacity is 100 requests per second and say million requests per second are coming into the system, then what? Of course, your throughput of the system will always be flat it cannot increase beyond capacity if anything if things are getting very, very your system is crashing and bad things are happening throughput may even fall but it will never go beyond your capacity.

So, the throughput flattens that is given, what else happens at overload? Your response time keeps on increasing whatever is your bottleneck component, the queues build up, build up, build up because, you have so many requests coming in and this guy is only doing 100 requests per second. So, this queue keeps on building up and therefore, every new request is having to wait a long time and it is getting queued. Now, when this queues are building up, what can happen, you can start seeing error messages.

So, if this is your web server, this web server has a lot of queue getting built up, and it realizes it has no time to process these requests. Then in such cases, HTTP servers can send a special response back saying service unavailable, if this web server sees that, I cannot handle

all of this requests, there is no CPU time available no resources available, it can send back an error message saying, sorry, come back later. Sometimes the application does not even have time to politely send you an error message like this.

Sometimes the application is not even you have your socket buffer and packets are sitting in the socket buffer, the application has not even had the time to read the packets and send a polite response back. So, the packets are just lying in some queue somewhere not getting processed not getting acknowledgements, and your TCP will timeout, connection will fail, system calls will fail. So, you will start to see even more worse errors.

So, this error is at least good. The application at least has enough time to say sorry to you. And even under if even more overload occurs, the application will not even have time, it will not even have time to read a request and send a sorry, response back. TCP itself will fail at the kernel level itself, failures will occur the system calls will fail or some switch some network card, some device driver, some queue will overflow packets will start to get dropped, all sorts of bad things will happen.

So, all of this is what you see as server overload when, say a big sale opens on an E-commerce website, or when a lot of people are booking tickets at the same time on a travel reservation website, any of these cases, so much load is coming in that your server is not even able to handle it does not even able to send back a sorry, response to.

In such cases, you are at your browser level itself, whenever you try to write or read from the socket, errors will occur, the connection itself has failed. And you will see this as a server crash. So, this is the result of overload very high response times, if at all your request complete, of course, if no errors occur in your request complete, you are gone to see very high response time because everywhere you are waiting in queues.

And sometimes you are not even this lucky to get response after a long time, you will get no response at all, and you will see various errors. So, this is what happens under overload. And the reason for all of this is simply the queue building up at the bottleneck component and the system not able to clear out this queue fast enough.

(Refer Slide Time: 18:56)

The slide is titled "Understanding saturation" in red. To the right of the title is a hand-drawn graph in red ink. The graph has a vertical axis labeled "put" (likely throughput) and a horizontal axis labeled "load". The curve starts at the origin, rises linearly, then levels off into a horizontal line, and finally drops sharply. A red arrow points to the peak of the curve, and another red arrow points to the sharp drop-off. Below the graph is a list of bullet points:

- Ideal operating point of a system: just below max capacity or saturation
 - Close to max throughput, not too long response times, no errors
- At saturation, some hardware resource at bottleneck is fully (100%) utilized
 - E.g., all CPU cores are fully busy with no spare CPU cycles
 - E.g., hard disk is running at full capacity performing reads/writes
- How to improve capacity?
 - Increase hardware resources at bottleneck component
 - Or, optimize system to use hardware resources more efficiently
- Sometimes, bottleneck due to software issues only (no hardware resource is fully utilized)
 - E.g., maximum number of file descriptors opened by process, cannot open any more
 - E.g., threads wasting time waiting for locks, even though CPU is free
 - Such issues can be fixed by rewriting code or tuning OS parameters

At the bottom left of the slide is the NPTEL logo. To the right of the bullet points, there are four red curly braces "}" stacked vertically.

So, given all of this the ideal operating point, if you have a system, where do you want to operate it, you want to operate it just below the max capacity. Your system throughput is increasing, increasing and reaching a flat point you want to operate, if this is your throughput, and this is your load. You want to operate your system, just about here, you do not want to operate it here because there is so much more work the system can do, you are not using it, but you do not want to operate it here also why because then there will be lot of overload and errors and all of that.

So, just below just that the max capacity. This is called the saturation point. And at that point also your response times are also low and as you reach the saturation point your response time start to increase a lot. So, before the response times increase too much. And when your throughput is high enough that is where that is the amount of load ideally you want to your system. And once you hit saturation what happens at the bottleneck, some hardware resource will be fully 100 percent utilized.

For example, if the component is doing a lot of CPU compared for each request, maybe all CPU cores have reached 100 percent utilization, there is no spare CPU available to do more work or if the system is doing a lot of IO the hard disk is running at full capacity. So, some hardware resource will be fully utilized because if it were not fully utilized, the system could have done more work. Now, if the CPU was free, the system could have handled more requests, why is it hitting a bottleneck because some resource is fully consumed?

So, how do you improve system capacity, you either increase hardware resources, if you have 4 CPU cores give 8 CPU cores then you will be able to do more work, your capacity will increase or given the amount of resources optimize your system. Write better algorithms that do not consume as much CPU for example, cache things, so that you do not use the disk so much for example, you can do all of these optimizations. These are the only two ways in which you can improve capacity.

And one thing to keep in mind is sometimes this bottleneck can also happen due to a software issue, no hardware resources fully utilized, but still, so, you see that your system throughput is kind of flattening out, but if you see the CPU utilization, utilization of any hardware resource, it is not fully utilized, it is not at 100 percent. Then what is happening, why is your system throughput flattening out, it could be due to some software issues.

For example, your process can open only say 1000 files and it has opened it is trying to open more than that, then it cannot do more work than that then the OS will stop, number of open files, number of file descriptors or threads are all waiting for a lock you have 4 CPU cores and but all the threads want only one lock.

So, only one of them can run on the 4 CPU cores. So, things like that, there could be some design issues, some software issues, some system configuration limits in your system due to which even though all the hardware resources are available, you are not able your throughput is still flattening out you are not able to do more work even if more load is offered to you. So, such issues, they should be fixed by either rewriting the code or tuning the OS and all of that.

So, if your performance is limited by a software issue, then you should tweak the software in order to make this issue go away. But if your performance is limited by a hardware issue, then then you either upgrade your hardware or do other things. So, in this way, whenever your performance is kind of not increasing beyond a point you have to see what exactly is the issue there.

(Refer Slide Time: 22:34)

Performance: parameters and metrics

- Given a computer system (with certain capacity and configuration of various components), how to measure its performance?
- Input parameters on which performance depends / incoming load
 - Number of concurrent users/requests in the system
 - Rate (requests/sec) of incoming traffic
 - Mix of various types of requests (which require different amounts of work)
- Performance metrics / outputs measured
 - Throughput of the system (averaged over a time window, end-to-end and per-hop)
 - Response time or latency (averaged, end-to-end and per-hop)
 - Utilization of various resources at components, especially bottleneck (averaged)
 - Various kinds of errors and failures (counts)
- Load testing of a system: vary incoming load, measure performance

So, now, this is a summary of what we have seen given a computer system, system will have certain capacity and each component will have certain configurations, it will have some service demands, which will decide the capacity or given the system what you will do is how do you measure performance you will vary certain input parameters on which performance depends for example, you will increase the number of concurrent users, you will increase the rate of requests coming in, you will change the mix of various type of requests some easy requests more hard requests, you will do all of this.

You will change the load, the incoming load into the system you will change. You will change the load and then you will measure various performance metrics. For example, you will measure throughput as you increase the load the throughput increases flattens out we have seen that, so, you will measure throughput, you will measure response time. The other metrics you will measure as you measure the utilization of the resources at the component especially the bottleneck component.

Why because you want to see as my performance is kind of hitting this plateau as throughput is flattening out is some resource at the bottleneck has it reached 100 percent utilization if not, why is my performance still flattening out all of these things you want to analyze. So, these are all the things you will measure, throughput response time utilization, of course errors, when overload is occurring, what are the errors, if there is no overload, why are still errors occur, all of these things you will measure.

So, this whole process of varying input and measuring the output metrics this is called load testing of a system. So, once you have built a system, you will do a load test you will vary input parameters, which determine performance and you will measure the various performance metrics. So, one thing I would like to point out is throughput, all of these things are usually averaged out because at any one second your throughput might be 105 requests per second.

At some other second it can be 95 requests per second, so, it can be slightly varying. So, what you will do is, you will do an average over some time window. And you can also measure for each link, this front end to app server or app server to database on each link you can measure latencies, throughput, errors everything you can also measure end to end. So, these are the various things, different metrics and different ways you can measure and all of this is called load testing of a system.

So, once you build a system, you will give varying amount of input load to the system and measure the throughput latency and various other performance metrics. Now, there are 2 ways of doing the load test to a system there are of course, many different variations these are roughly the 2 ways.

(Refer Slide Time: 25:21)

Types of load testing

- Different types of load tests based on which input parameter is varied
- **Open loop load testing:** vary rate of incoming traffic into system
 - Generate N req/s for increasing values of N (mix of requests can also be varied)
 - Can be implemented by firing a request every $1/N$ second
- **Closed loop load testing:** vary number of concurrent users of system
 - N concurrent users, each user sends a request, gets response, waits for some amount of time ("think time"), sends next request
 - Can be implemented by having N threads/processes emulating N concurrent users
- Both techniques are valid ways of varying input load of system
 - Open loop testing can lead to higher number of concurrent users, more queueing
- **Load generators:** software programs or hardware appliances that generate load to test a computer system in open/closed loop manner
 - Provide knobs to vary rate of requests, or number of concurrent users etc.

One is what is called open loop load testing that is here you vary the rate of traffic if you have a system you give it say 10 requests per second then later on you give it 100 requests per second, then later on you give it 500 requests per second. You vary the rate of incoming traffic into your system that is called an open loop load test. The other load test is what is

called a closed loop load test that is you vary the number of concurrent users who are using the system the number of concurrent users you vary.

So, there is no one right way to do a load test, these are both different ways in which you can increase load vary load into your system. So, how do you do an open loop load test, you will pick some value of the rate of requests into the system and you will generate if you say you want to generate 10 requests per second you will pump in that much load into the system for increasing values of n you might start small keep on increasing rate of requests and see where does the throughput flatten out and so.

And how do you implement it, you can simply have a software program that fires a request every $1/n$ of a second if you want to send 10 requests per second every 100 milliseconds or so, or with some randomization you can send these requests. And in a closed loop test you vary the number of concurrent users and each user will send a request get a response then typically wait for some time.

So, you have a think time. The user will send a request, get a response, then think for some time, pause for some time, then send the next request. That is how usually closed loop testing is done. In order to emulate this is what real users would do if there are real users in a system this is how they would do they would send a request, get a response, think, send the next request. So, such users you will emulate some n of these users into the system.

And you can write a program to do this you can have n threads, which is each making a request getting a response thinking and so on. So, both these are valid ways of varying the input load into a system and not that open loop testing is usually leads to higher number of concurrent users because so much load you can easily increase the load overload the system more easily because, so many requests are coming in. And whereas here you have some limit, there are only some n concurrent users, there is a limit to how much load can come in.

So, usually closed loop load tests are slightly easier on the system, they do not crash the system as often but both of these are valid ways of testing the system. So, in general, for any system, you have certain specific pieces of software or even hardware you can get a hardware box whose only purpose is to generate load these are called load generators. So, where you get a program or you get along with the hardware everything, where you connect it to your system, its only job is to just generate load either in an open loop fashion or a closed loop fashion.

And you can connect this to your system and you can see you can do a performance test how is your system performed and these load generators, they will provide various knobs to you, you can vary the rate of requests if it is an open loop load generator, number of concurrent users, think time, the mix of the different kinds of requests all of these you can configure and you can send in a certain amount of load into your system.

(Refer Slide Time: 28:49)

Running a load test

Load gen → System

- How to run a load test
 - Setup load generator, system under test and connect them suitably
 - Generate increasing amounts of load from load generator to system (by varying rate of incoming traffic or number of concurrent requests)
 - Measure output metrics (throughput, response time, errors, utilizations) for each value of input load
 - Eliminate sources of non-determinism (e.g., pin threads to CPU cores)
 - Ensure load generator or connecting network is not the bottleneck
- Results of load test: performance metrics vs. incoming load
- What after load test? Performance analysis and engineering
 - Analyze and understand performance metrics, identify bottleneck
 - Optimize system, repeat load test
 - Stop when system performance meets the expected incoming load

NPTEL

So, finally, how do you run a load test, given that you have built a system you will do what is called a load test to your system that is here is your system under test. It has many components it is deployed somewhere and then you have a load generator. This is either this software program running on a separate server or it is a hardware box itself that generates load, you will connect these two up, the load generator will keep sending load into your system incoming load at some varying rate and your system will handle that load and then you will measure various performance metrics.

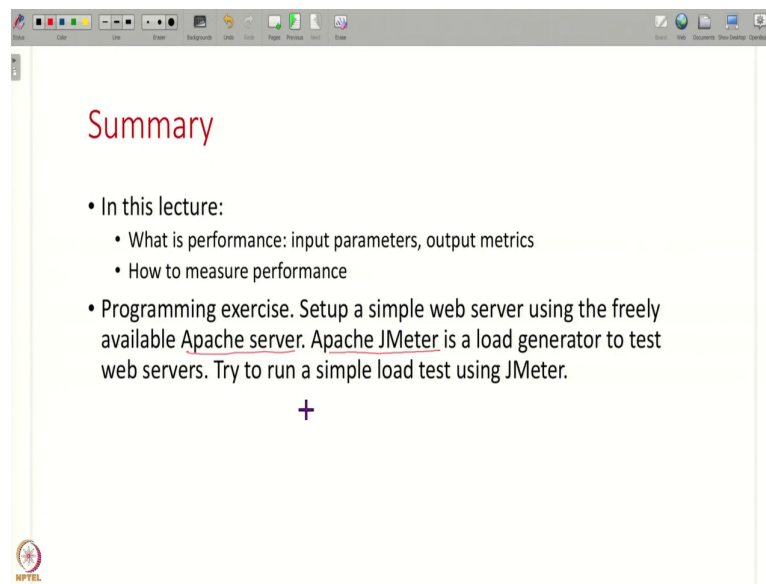
You will see what is the throughput, what is the latency errors, utilizations all of these you will measure and you will plot some graphs. These performance metrics versus incoming load you will plot the graphs that is the result of your load test. Now of course while doing load test, you want to take care for example, you want to ensure that your load generator is not the bottleneck. If your system can handle 1000 requests per second, but your load generator is only generating 500 requests per second then it is not useful. No, it is not good enough to test the performance of your system completely.

So, you have to ensure that the load generator or this network that is connecting them or none of these are the bottleneck you want your system to be fully loaded. Therefore, the bottleneck should be here you have to take care of that and you also usually want to eliminate sources of non-determinism because, for example, Oh, this thread was running on this core then it performed differently oh no all of these threads had some other behavior in this time. Therefore, they performed badly you do not want to give such excuses therefore, you want to have as much determinism as much reproducibility.

So, fix all the parameters, fix your the number of threads, number of CPU cores keep everything constant and run a load test, so, that you can fully and correctly measure the performance of your system. And after the load test, what do you do? You do you analyze your performance and then you do some optimizations, you do some performance engineering. So, first you will see what performance have I measured? Is it making sense or not, okay fine, it is making sense you identify a bottleneck, then you optimize the bottleneck, repeat the load test and you can keep on doing this this is an iterative process.

You keep on optimizing your system, again measuring performance, again see the results optimize, measure, optimize and so on. You keep on doing this till you are satisfied till you say fine, my system has good enough performance that I can deploy it on the field.

(Refer Slide Time: 31:29)



Summary

- In this lecture:
 - What is performance: input parameters, output metrics
 - How to measure performance
- Programming exercise. Setup a simple web server using the freely available Apache server. Apache JMeter is a load generator to test web servers. Try to run a simple load test using JMeter.

+

NPTEL

So, this is a summary of what we have seen in today's lecture, we have understood the basics of what is performance? What is the input load to your system that you can vary? And what is the performance metrics that you can observe about your system? And how do you measure performance? How do you run a load test? And what are the things you have to keep in mind when you are doing all of this, so the basic concepts pertaining to performance measurement we have covered in this lecture.

So, as a programming exercise, you can actually run a simple load test yourself, you can install the Apache web server, and there is also a load generator that can actually generate load to this web server that can actually at whatever rate you want, send HTTP request to this web server and measure the performance of the web server. So, you can try and use these tools on your own to understand what is a load generator and how to run a load test. So, that is all we have for this lecture. Let us continue this discussion in the next lecture. Thank you.