Performance evaluation of computer systems

Prof. Krishna Moorthy Sivalingam

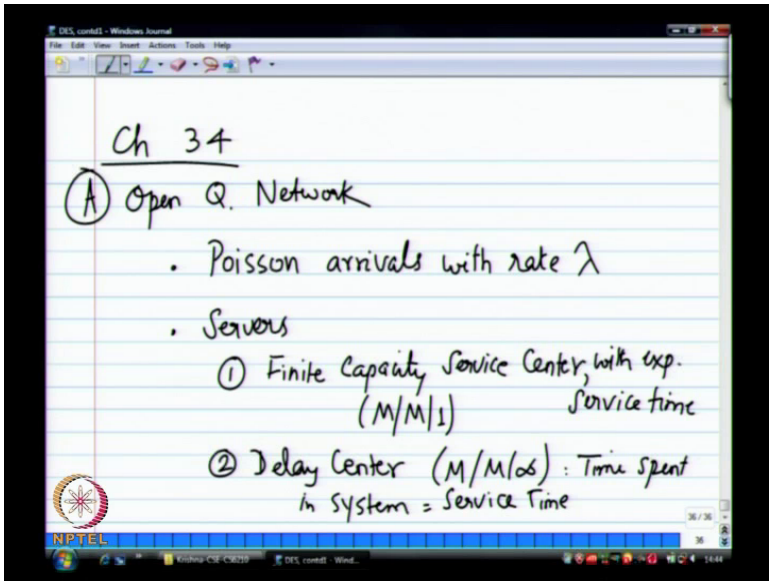
Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Open and closed queuing networks

Lecture No. # 26

(Refer Slide Time: 00:16)



Now, we move on to chapter 34, which is your basic mean value analysis (no audio from 00:17 to 00:27). So, we will first do analysis for open queuing right, an open network. It has a network of queues, but network is open; so, there is one input point, one exit point. So, we saw as to what can be open queuing a transaction processing system, there you go through multiple queues; transaction arrives gets process, process then leaves right.

Even our cpu system can also be looked at that, it is not necessary that it should be the closed (( )) with that as your cpu is there, devices are there, jobs keeps coming to the cpu, they get process they leave a system without having to the circulation business. In circulation, you already were assuming that there is always a replacement job for a given job that is completing. Open system job arrives finishes leaves right, just trying to find out the average

response time in the system, it depends on lambda; in this case, the closed system it depends on n, this is the (( )) mean system parameter.

So, the arrivals are assumed to be Poisson arrivals with rate lambda. And the servers are of two types, so one it is an so called finite capacity. This is a is a finite capacity service center which means, the service time on the service does not depend upon the number of customers in the queues, independent of the number of customers waiting in the queue, which is a M M 1 system basically right; so nothing to be, the example. So, this is with here we are going to look at exponential service time, but so this is the single server with exponential service time or standard.

All this the server could also be a simple delay center it m g infinity or M M infinity before right. So, called delay center. So, delay center is nothing but, device it introduces the constant amount of delay. So, there is no there is no queuing in that particular subsystem that you are enter, whatever you enter you always spent S i time units service time then leave that system right. So, there is no queuing at all that is the actual system.

From a queuing perspective we represented as a M M infinity right, where for a whenever a customer there is always a server created to service that customer and then you get serviced. So, there is no real upper limit. It is sort of like a web browser or web server, anytime has an http request you create span a new thread and service the new request right. In theory it does not infinite, but practically you make; theory it is infinite, but practically you might have some limits there, no more than a million threads or something like that.

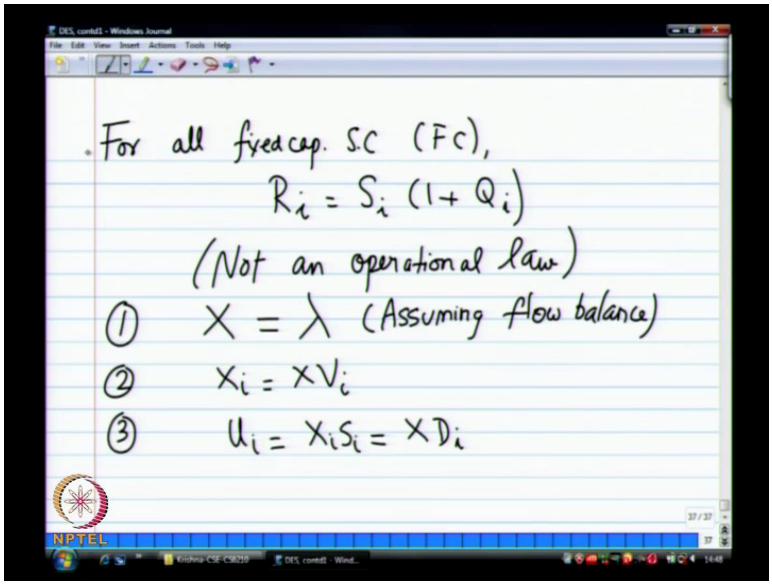
So, essentially a time spent in the system is simply right, so the time delay is the time spent service time and you can derive this very simply right. If you look at a M M M system, we have after some point the service time is simply m mu right, after m it is m mu m mu, but in this case it will be n if the system state then will be simply n mu. So, you can very easily derive that service time we can we can go for a final exam we can figure it out M M infinity. The results are there in the book the derivations (( )).

So, these are the two service centers, we are not looking at the so called M M M system. These are low dependant service centers, where depending on the number of customers in the queue, this service centers capacity will be increasing there is only one customer the service capacity is mu; there are two customers, service capacity is 2 mu, 3 3 mu and so on; therefore, that is variable that is called low dependant service center right.

See, that is that is a for chapter 36 that comes later we looking at a simpler system now. So, this is my specification I have a network of queues, no standard Poisson arrivals, but these are wait these low dependant service centers is not being considered here.

So, now based on what we saw in chapter 33 we just gonna have, if give you a queue of networks which is open you should be able to simply run through that like a machine; and say this is the delay, this is the average delay, this is the average queue length, average utilization, average throughput; all that is very simply computable you can write at 10 line program (( )) and that specifically what you are going to illustrate.

(Refer Slide Time: 05:47)



So, the only addition that you will see here is, for all fixed capacity service centers we will call this FC from (( )). This is the response time is simply that, what is that mean, do you know (( )) before right. So, the time spent the respond that total time spent in a Q i is simply just average service time which is S i into 1 plus Q i. So, when I arrive to the queue, I will find Q i customers waiting right, this e of n q at the time of arrival.

So, I have to wait until all these guys finished their service. This of course, requires some memory less property assumption that we are (( )), this is not an operational law, this is the convenience law right. So, therefore, there are Q i customer they are all take you into S i even though this guy met a partially finished service. I simply saying, when I arrive that Q i guys, the remaining service time is simply Q i, we did the residual service time analysis separately

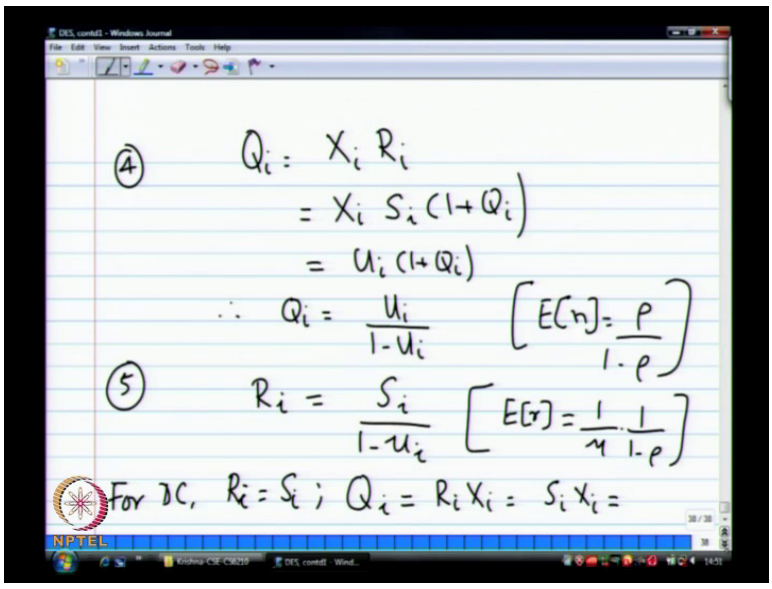
before; but in this case, we are assuming the all Q i customers will take an average S i time units.

So, I have my my total waiting total response time in these devices S i which is my own service time plus Q i into S i for all the other guys that is all. This is only another addition what we saw in the previous chapter, this is one extra rule. So, this is not an operational law. So, now we list a series of steps to sort of figure out all the performance metrics for this given open queuing we look an example, but the steps are as follows.

So, as the first thing is I am I will be given lambda for sure right, these are my inputs I will be given S i's, I will be given V i's in some form of the other whether these I cannot proceed further. Therefore, the system throughput assuming flow balance right simply lambda. So, this implies that the number jobs leaving an average equals to the number of jobs arrival.

So, my first remember everything depend on this X. So, we got X then, we can get our X i's, because we know V i. So, V i has to be given otherwise, we are look for the description, if it is not there then it is tough (( )). Similarly, you can measure something (( )) looking at logs, we can find out what the average (( )). Then we know that, u i specifically (No audio from 08:49 to 09:15).

(Refer Slide Time: 09:24)



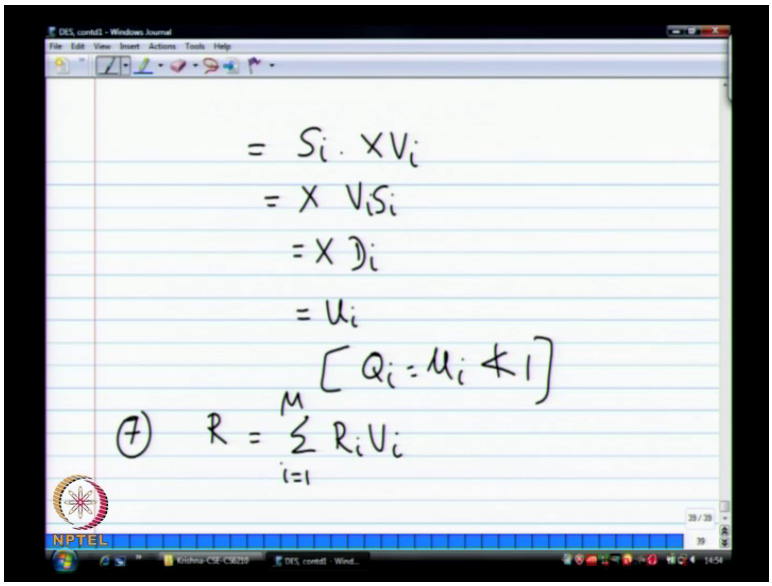
So, then after we need to find out this Q I; so, Q i is equal to X i R i Little's law right. And X i and R i be expressed that in terms of our assumption here that, R i equals S i into 1 plus Q i

(No audio from 09:41 to 09:49) which whether it happens to be u i. So, therefore, Q i gives u i by 1 minus u i, and this (( )) seen already right; this is E of n for the single queue rho by 1 minus rho derive same thing shown up here.

So, we know an expected number of customers in any queue is simply once you utilization of the queue it is simply rho by 1 minus rho, so same thing. So, basically I am getting my X, my u from u I can get the queue; from once I get the queue then, I can get my R. So, R i I can get R i whichever I want to; once I can use this expression right it is basically Q by X or I can also derive that to be and by the way right or 1 minus 1 over mu minus rho. It is a notational different unfortunately, but that is all right.

So, basically I can look at every queue (( )) all these values. So, once I get all of these and for delay centers, R i equals S i there is no queuing at all right (No audio from 11:37 to 11:50). And the number of customers queued in an M M infinity queue is simply rho, you go back and look at it will be just rho. So, again we can derive this if you want, this is R i into S i X i right, this is the formula as before; but R i we said is S i into X i which is I will go to the next page and this for R i equals to S i right. So, S into X is basically X is again X into V i right. I can go to the next page can I.

(Refer Slide Time: 12:33)



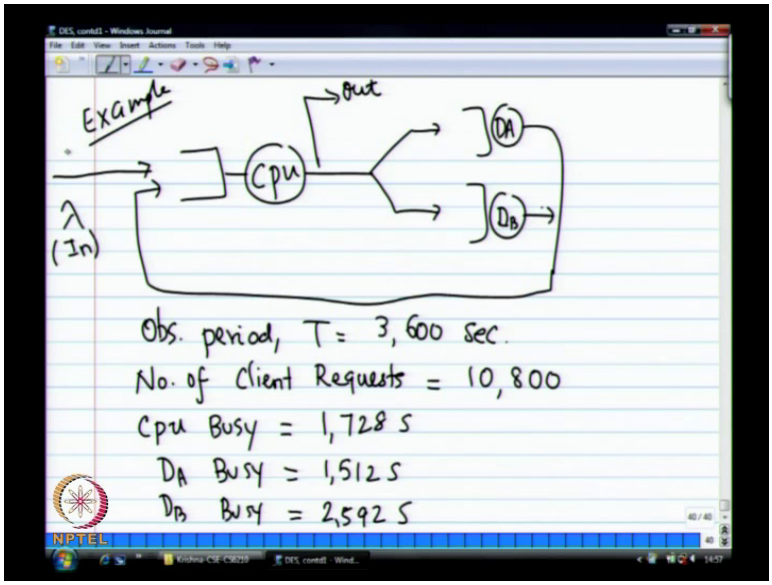
That equals yeah S i into X into V i. So, that is basically X V i S I; V i S i is nothing but, D I; and X D i is nothing but, u. So, queue number of customers queued is simply equal to the

utilization of the device. And M M infinity, mu i is need not be less than 1, because there is a that restriction is not there, we can have as many customers as you want, you will always have a server waiting to serve you.

So, therefore, this u i and other words, Q i equals mu i need not be less than 1. I can have 100 customers waiting in the queue right that restriction of rho less than 1 applies only for the M M 1 infinity, not M M infinity (No audio from 13:27 to 13:35). So, once you get all of these then, I can compute my R right; R is simply R i V I (No audio from 13:45 to 14:04). So, that is what I basically want to find out right, I know a throughput I know a response time.

So, this you can very well right implement as a next assignment open queuing network you know how to connect you know how to do a single queue connect the queues feeding traffic; we can then verify whether all of this is correct right, all whether the mean value analysis is correct or not. So, that could have been your project 3, but it is not; I figure that close queuing is more interesting than open source, if you want you can do open source same thing right it just slight modifications, but you gonna do open queuing networks; those are you finished project 2 that will be your assignment its almost (( )).

(Refer Slide Time: 14:52)

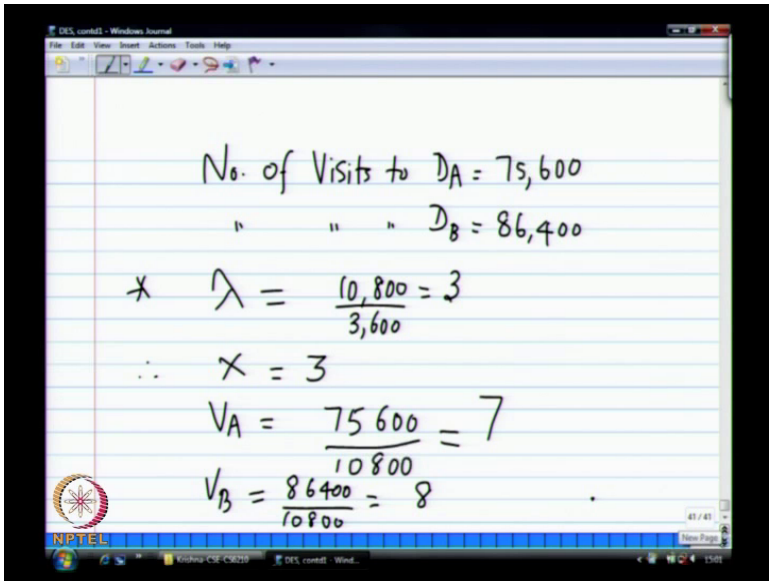


So, let us look at an example. They being going for almost 15 minutes with only X i's and u i's let us figure that (No audio from 15:10 to 15:57). So, that is my classic (( )) right, 2 disks 1

cpu (No audio from 16:02 to 16:21); and occasionally cpu finishes the job in the process leaves the system.

So, now for the input specification; so, somebody has a system administrators given you a logs and you have done some measurements right. So, the observation period, T is 3600 seconds, so you have taken a 1 hour (( )) of data. So, the number of client requests or job requests right that you read from the log file 10800; and then every time right, the log will tell you cpu is getting used, cpu is starting use somebody (( )) start of cpu utilization, end of cpu utilization. See, you are measuring this cpu utilization time right. So, the busy time for the cpu was 1728 seconds then, your disk A busy time for 15152 seconds; and the disk B (No audio from 17:43 to 18:06).

(Refer Slide Time: 18:14)



Then, the system also right from the log files you found that the number of completions or visits right to disk A, total number of visits by all the jobs was 75600. Number of visits to D B was 86400 this is what we are given. Anything else we need? No. So, with this you given a network you are given some of this operational values measure, now the goal is to find out right (( )).

And the goal is not just to find out the response time, goal is to use this to find out where the bottleneck is and then try to enhance your system performance by removing bottlenecks or making some disks faster and things like that that is the whole process. So, if it is today 36

there are 10000 jobs per unit time, what happen if it is 20000 jobs or 60000 jobs arriving then, how do you use these values to extrapolate and then find out what will be future delay right. So, that is the whole idea of this performance evaluation.

So, how do we start, where do we start, X start with X, before x we need lambda. So, all this lambda is a number of jobs per unit time, what is that? So, therefore, so X equals 3. So, what is next, what can I figure out?

X i

X i, for X i I need V A, what is V A?

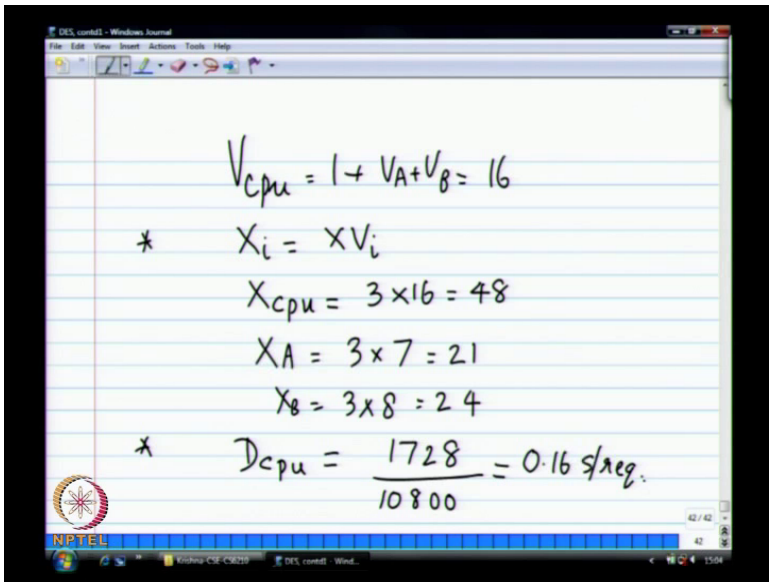
(( ))

So, we know the right, so V A the number of visits, how many visits we are made by the jobs, there are total 75600 requests to disk A and there were 10800 jobs that finished right. So, this assuming that (( )) no job at time 0 and no job at time 3600 right; it is possible that some jobs will working we just ignoring all that, we are assuming inflow balance that whatever started is gone.

Otherwise, this could be some other job that is started even earlier the (( )) (( )) that still going on the system. So, there are some there of convenient assumptions we are making by ignoring all those older jobs. So, we find there are 75600 requests totally made only by these 10800 jobs that completed; therefore, these jobs must have requested an average 7 right. These also average we do not know every job requested 7 we just saying that, average is 7.

And next one is V B, so this is again conveniently given no need for a calculator its 8 right; we said, V B had this (( )) had 86000 request 400 request, V B equals 8. And we know that, every time you go to your disk you have to go to the cpu right we saw this before. So, V cpu is simply 1 plus and one final execution, where there is no disk access required, it is simply computing it the process finishes that is the outline right.

(Refer Slide Time: 22:06)



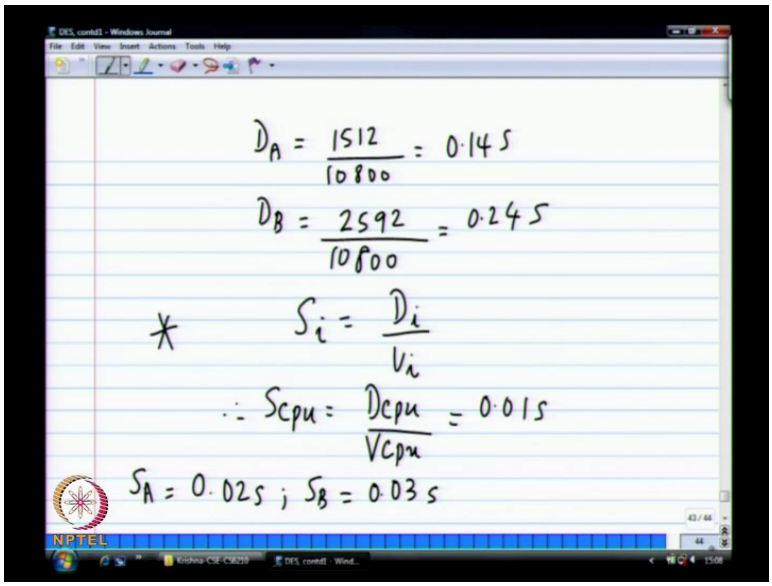
So, V cpu 16, so now we can do all the X i's. So, now we can look at the throughput of every device, so throughput of each device then leave as to other things. So, X cpu is, so the cpu handles 48 request per second right and this disk handle (No audio from 23:00 to 23:13).

(( )

Do fast, do fast. So, always do this, (No audio from 23:30 to 23:42), now what we need service times. So, what is the service time, average service time spend at the cpu disk and so on, available. So, what is the average service time for the cpu? You have the busy times right, from the busy times we can figure out. So, you know that, he total busy time was 1728 divided by the number of jobs that completed right.

So, we can actually compute the D cpu first, I know my D cpu right. So, D cpu is what as actually given, the total demand on the cpu (No audio from 24:31 to 24:40). So, D cpu total time the cpu as busy is basically the demand right made on the cpu (( )). That is totally it is busy for 1728 units of time seconds and the total jobs it finished were 10800 right. So, each job an average the total demand made by the job is (No audio from 25:01 to 25:19).

(Refer Slide Time: 25:20)



(No audio from 25:21 to 25:30) So, D A is a 0.14 and then D B is 0.24. Now, it is better to verify these values. So, the book is got a huge error list of errata. So, this (( )) (( )) demands for each of the devices. So, once I know the demands, how do I get the service times? D equals V i S i right, D equals V i into S i; total demand is a number of visits average visits per job into the average service time. So, S i equals D i by V i.

So, therefore, S cpu (No audio from 26:17 to 26:28). So, the service time for that is 0.02 seconds, service time for A is also 0.02 seconds and service time for B is 0.03 seconds.

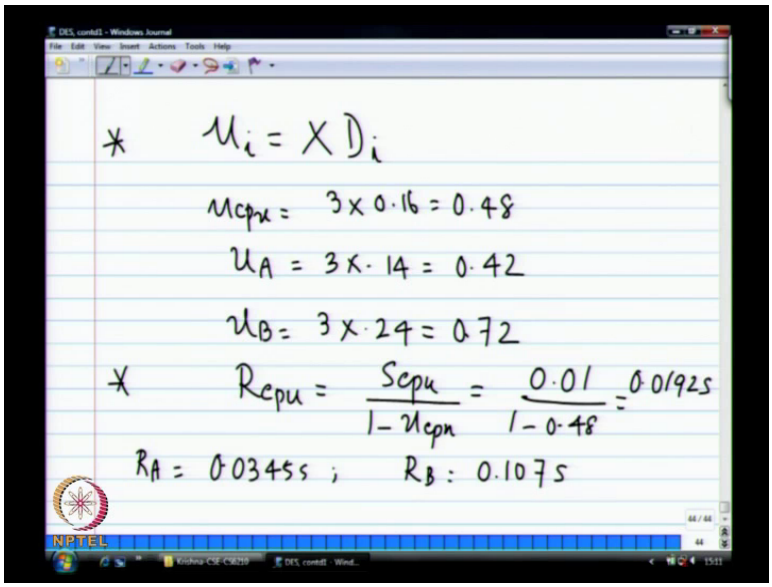
(( ))

Sorry, it is not the books fault its I say (( )) I wrote my 2 as 1. (No audio from 27:01 to 27:10) So, which is the bottleneck device this B that is taking maximum service time, maximum demand is B. So, if you want to reduce that service that, either service time as what I can do, so or I can even reduce the number of visits.

So, how will I do that, I (( )) faster disk or number of visits how do I reduce between the two disks; you have to basically move some file system right some part of the file system to disk (( )) faster that way we reduce the number of visits disks (( )), because some files are sitting there, that accessing a lot, but unfortunately the disk is slow. So, you should try to basically balance your file system such that you essentially come up with almost equal right (( )).

So, now we are found s also; now I once I find s then, I can find my I do not really need the Q i's right. Because once I get the service time well I need the utilizations (No audio from 28:16 to 28:25) sorry we need u i before that.

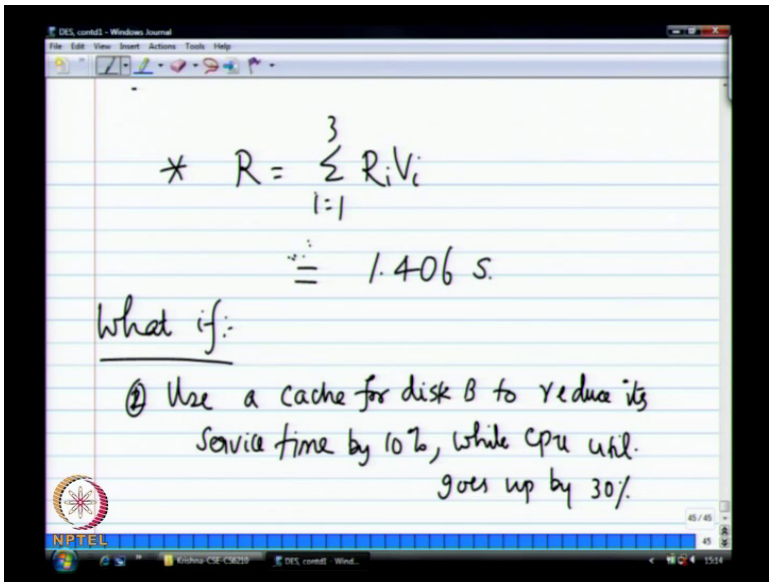
(Refer Slide Time: 28:33)



So, then we will find out the u i. And u i is given by X D i right. So, since I know D i, I know X; so, therefore, u cpu the utilization of the cpu is 3 into 0.2 0.18 0.48. Utilization of A is 3 into 0.14 that is 0.42; and u B is 3 into 0.72, again check right. When you come to this stage in your exam, you find that u i greater than 1 you know that somewhere that you went wrong. All u should be less than 1 and also the highest utilization is for disk B that is a bottleneck right. So, you just take a moment to check those.

Now, now I can use my standard formula. So, this is s by 1 minus u cpu, this again 1 minus 1 by rho 1 by mu into 1 over 1 minus rho are standard M M 1 formula this showing (( )); service time by the time that the server is actually idle. So, that is now 0.01 1 minus 0.48. So, this is 0.0192 seconds, R A likewise 0.0345 and R B is 0.107 again check that, disk B is got the highest response time (( )) (No audio from 30:32 to 30:54).

(Refer Slide Time: 30:55)



And then I can compute my overall system response time per job is using all these values derived so far. So, what is R what is the average response time with we have no everything at hand, the last formula (No audio from 31:32 to 31:57). Now, with this log file we have been able to translate this into this set of numbers. Now, if your boss says right, so what if boss is not happy with 1.4 seconds; and it is a too much response time, you want to reduce the response time.

So, the boss says do something about it, we say one use

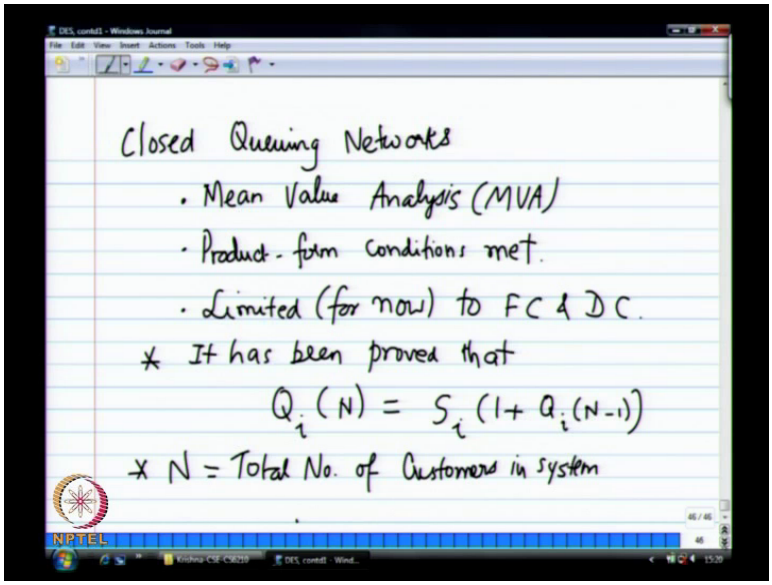
(( ))

So, this (( )) option use a cache for disk B is to reduce the service time. But if I am going to cache that means I am going to have more cpu intervention. So therefore, while the cpu utilization, this is again guess on your part right, while cpu utilization goes up by 30 percent. Basically, our cpu response time right, service time will be higher for the cpu (( )). So, (( )) look at some sort of compromise.

So, now you do not have to go actually build a system right that is go and have that kind of 10 percent; simply plug this numbers, you can now count the approximate idea for the (( )) then, you can like a intelligent choice has to whether to this is what. So, that is where your analysis techniques come in handy. That is your analysis for a complete queuing system.

So, given any open queuing network with the only one entry point in the simply do that. (( )) open queuing network has multiple entry points as in a packet right several routers some packets getting injected here; then also along the way you can get new packets add it that every part of the system. That is also an open queuing network and multiple exit points. Now, we need not worry about that now, we will leave that some other class. So, this is the single input single output system. (No audio from 34:06 to 34:18) So, questions on this so far (No audio from 34:24 to 34:31) none. So, that is the open network. Now, let us go to queuing network, closed networks right.

(Refer Slide Time: 34:41)



So, for this the technique used is what we called as a mean value analysis, so closed queuing networks. So, in the closed queuing networks your input is no longer lambda it is a n in everything else it's similar to the other system. So, we use mean value analysis. So, this will be applicable only for the systems that meet the condition for product-form solution, initially it was just the M M systems.

But, then later on lot more systems that have been listed I did not just mention those to you (( )) right B C M. P. Chandy and others paper 75 that should lot of conditions. So, there are more generic more general networks that can be looked into we are just making this assumption right only for the… So, here again we are only looking at (No audio from 35:52 to 36:00) the two systems where there is a fixed service or fixed capacity and delay center.

So, I am not considering M M M system just M M 1 system and delay centers, M M infinity systems. Now, let us look at what is involved in actually try to find this. So, analysis is more or less similar to what we saw before it is an iterative process. So, the system (( )) (( )) system we start with just one job, it enter system it goes device by device it finishes then it then replaces the next job, when the first job is complete. Then you inject (( )) this is we saw what of (( )) is right will be d and so on.

So, you inject one more job and see what happens to the queuing status right. And then I keep on increasing value of n until you find out some of optimal value for n. So, to do this analysis, so you will be given n then, you want to find out the response time with n users and the throughput with n users; R of N, X of N is what you are looking for n is input, S i V i all those things are given to you .

So, the basic premise that it (( )) is right this was actually shown right this is not a guess. So, to shown that and we are not really worrying about it, where this queue is an important. The number of elements queued in a device is what you want to know, once you get the queue then I can derive many other things around it.

In the last open form open network it was a Q i equals S i into 1 plus Q i that is known, but in this case I have similar formula right. So, the number of customers queued in device I, when there are n customers circulating in the systems is equal to S i 1 plus Q i then something else (( )) (Refer Slide Time: 38:07). So, when n job arrives right, the delay it faces is the service time for itself plus the number of customers queued ahead of it, when they were N minus 1 job in the systems. That is that is the basic result that we use, they are proving it we just taking that as matter of fact.

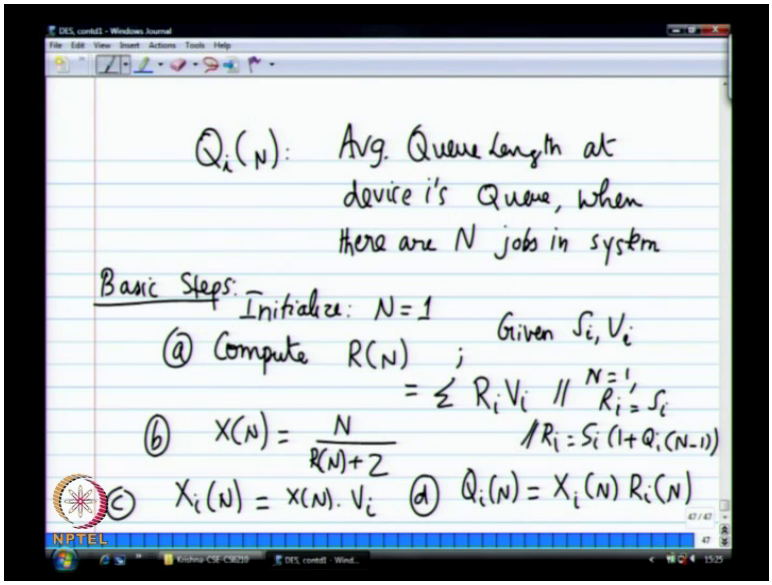
So, the previous expression was S i R i equals S i into 1 plus Q i, here it simply S i into 1 plus Q i N minus 1. So, to compute values for R of N, X of N I need to know the corresponding values for N minus 1 it necessitates becomes a recursive process right, so from 1 2 3 4 and so on.

(( ))

N is a number of jobs in the entire system, the number of jobs which is distributed across these several different queues. So, so let me put what this N is, N is the total; when there are

N customers, the length of a Q i length of i'th device queue that the notation that is that is the notation that we are using I will write that down. I put that in next slide.

(Refer Slide Time: 39:27)



So, Q i of N is the number of or is the average queue length at device that is i, i's queue, when there are N jobs in the system. So, if my degree of multi programming is N then, this is the number of customers I will see at jobs queued at at i'th device. If I know the number of jobs queued at device i, when there are N minus 1 jobs in circulation using that I can find out the number of jobs, when there are N jobs in the system. That is the formula that are you going to use; (( )) it is exactly similar the other one right, there are N minus 1 guys there were so many jobs queued.

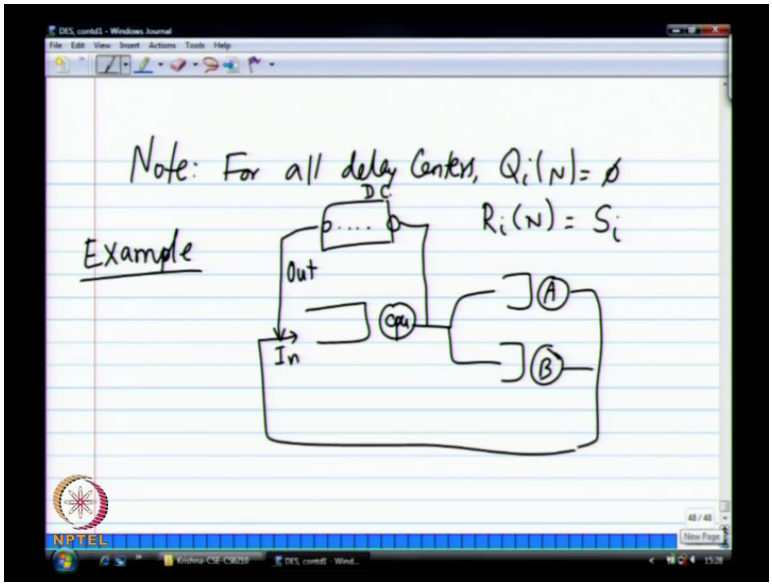
Then, when n eth guy comes right this is the this is again we can pretend this is true right we just logically derive that should be like we did in the other case; but, actually proved that, that is done in some 1980 paper, which we want the time to go through now, now, the steps for MVA; so, the basic steps right. So, initialize N equals 1. So, one customer you can always find out, we know what the R is, we know what the D, R is going to be simply D.

So, compute R of N. So, we assumes that, we are given S i and V i that is pretty much all that we need right, service time plus number of visit to each center that is about it. And then from that right, we know what R is, R i in this case is simply equal to the service time, because there is no queue in the system at all right, your Q i of 0 equals 0. So, when you start with

that. So, I can simply compute that, if there is no queuing, everybody is only job flooding arounds. So, Q i of i equal of in 0 equal 0 therefore, I can easily compute R of N right. This is from my R i V i and initially for N equals 1, R i equals S i. So, then when I compute this, now this is the time sharing system right. So, we have this think time also introduced. So, I can start with computing R, compute throughput, because we know the response time then, I can compute the each device.

And once I compute X, I can compute Q. So, Q i of, so what is Q equal to X i (( )) right. So, this is simply X i with N R i with. So, that completes one step of the iteration; starts with 1 N equals 1 compute R then rest of it will naturally follows. Now, I have basically compute my Q i of N. So, now I go back and then I have to compute R of i right, I will be using my approximation that is your mean value analysis. So, given a value of N you simply have to run this N times to find out (( )) delay center R i is always S i.

(Refer Slide Time: 44:05)



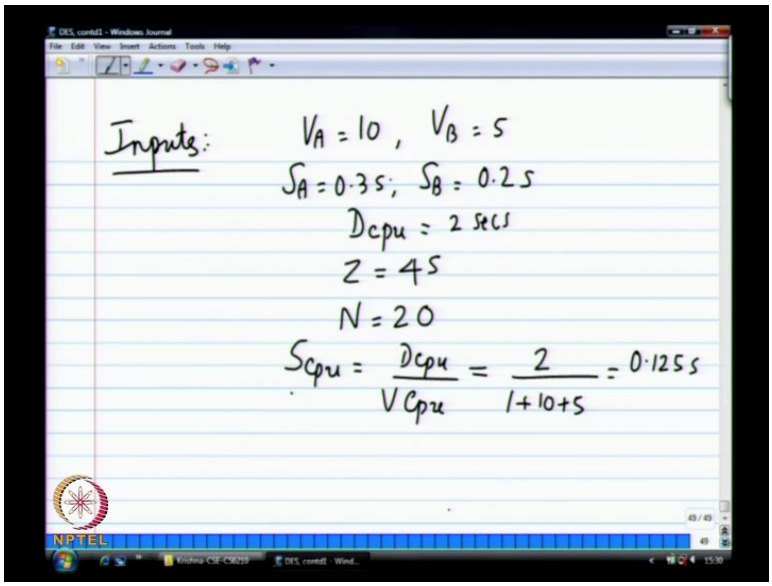
Note, for all delay centers, Q i of N equals 0; and R i regardless of the number of customers is S i. That is your end of MVA, the book actually gives your algorithmic description, but that is basically the same thing, if you implement you simply use that description (No audio from 44:35 to 44:48). So, now let us look at that same example right.

So, this is now that cpu then, it goes to one of two disks (Refer Slide Time: 45:13) (No audio from 45:14 to 45:22). Then now and then the job finishes this is the interactive system. So,

there is a delay center right, which has a infinite, this is a delay center that is D C there infinite. So, the delay center is where your interactive terminal is (( )) and thinking and then after thinking it comes back a joins a queue. So, this is technically our inline and this technically our outline.

So, whenever a job is coming in from here to here which means that, that job is basically completed right and then it's a new job that is replacing. So, we measure that the job traverse in this link the out link that is where we would say that, that is your throughput is measured. This is what we are seen before. So, now in this system we we have to find out right, what is the throughput of the system has an increases.

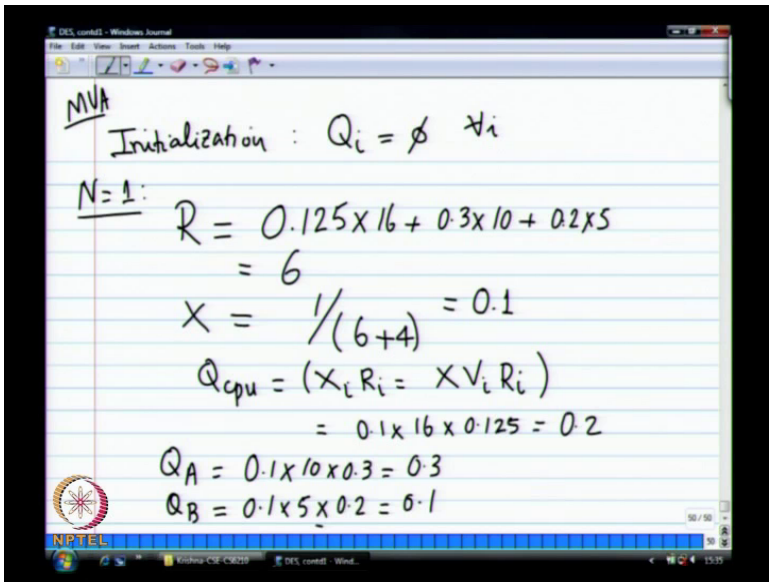
(Refer Slide Time: 46:38)



So, the input right that is given to us that V A is 10, V B is 5 and the service time of the first device is 0.3 seconds, this is different from the just previous example that we saw; so, do not confused. So, given s and V, then we will change the also giving us D for the cpu and that Z is 4, N equals 20. So, with this scenario, what is the average number of jobs completing and how long does each job take to finish the system.

So, we need to find out what, so where do we start (( )) I know D cpu, I should find out S cpu right, service time for the cpu, because that is what I am using in my calculation. So, S cpu equals D cpu by V cpu, which is 2 divided by 1 plus 10 plus 5, so there is 1 by 8; so, its 0.125.(No audio from 48:17 to 48:25) So, we are pretty much (( )) (( )) questions.

(Refer Slide Time: 48:48)

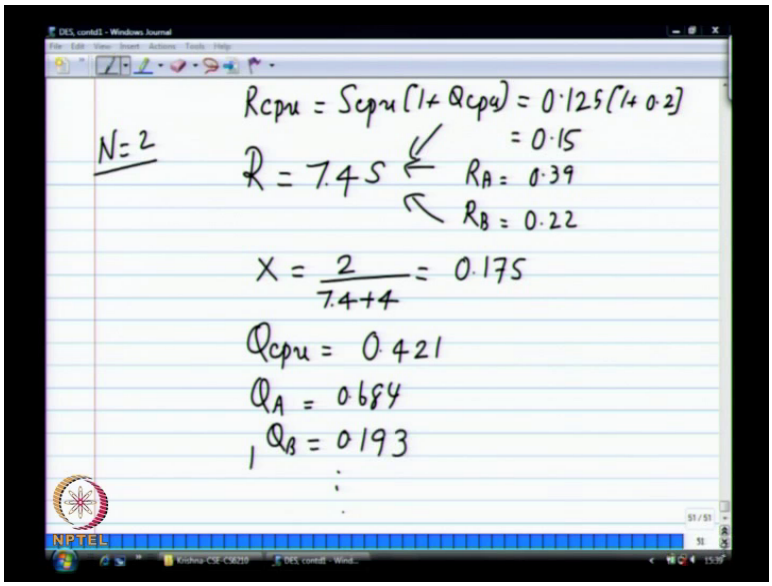


So, initialization, so this is our execution of MVA. We just said, Q i of Q i equal 0 for all i right then, this is as N equals 1. So, we simply have to find out all the R's right R i's get the R, (( )) someone wait and watch and you tell me the values. So, what is gonna be x (No audio from 49:27 to 49:37) no queuing right therefore, R i equals S i (No audio from 49:42 to 50:08). So, response time is (No audio from 50:10 to 50:23) (( )) plus 2 plus 3 plus 1 plus 2. So, and then X is 6 by sorry N equals 1, R is this, Z is 4, 0.1.

So, only one job in the system my my average throughput is 0.1, because each job take each job takes 10 seconds right throughput is just 0.1 then, we want to compute the Q's right. So, Q cpu is now equal to say Q equals X R, X (( )) know, R I know yeah X i R i sorry X i R i (No audio from 51:38 to 51:46) and we can we all know that this is true right. So, that is 0.1 into 16 into 0.125, so this so the new value for Q cpu which was 0 as now equal to 0.2.

And the value for this one is equal to (No audio from 52:15 to 52:36), so that is the value. End of first iteration I have new values for Q (No audio from 52:41 to 52:52); I again as you check you should note that, the queue length should always be less than or equal to, what should the queue length always be bounded by 1 or queue length? By n right, because we can add more than n customers in any queue, total capacity system is that much. So, but (( )) some of these queues is not equal to 1 I have only one customers circulating 0.2 plus 0.3 plus 0.1. So, what happen to other 0.4 delay center right thinking time.

(Refer Slide Time: 53:50)



So, that is what it is capturing. So, now we can sort of mechanically run these things right, write a small program in simply just compute all of this left and right. So, if you want I can put in step by step otherwise, let you go compute the step by step and I just give you the numbers, thus we go along right, because you are more interested in.

(( ))

In the first step, so this is

(( ))

That customer is in there delay center, because before you start the guys

(( ))

All the queues are empty, when we start the system these three queues what we are looking at right and the delay center there is no queue as such right. Customer arrives always there is service time, except that when you finally, finish and go to the delay center you will spend a constant amount time before coming back to the system as such.

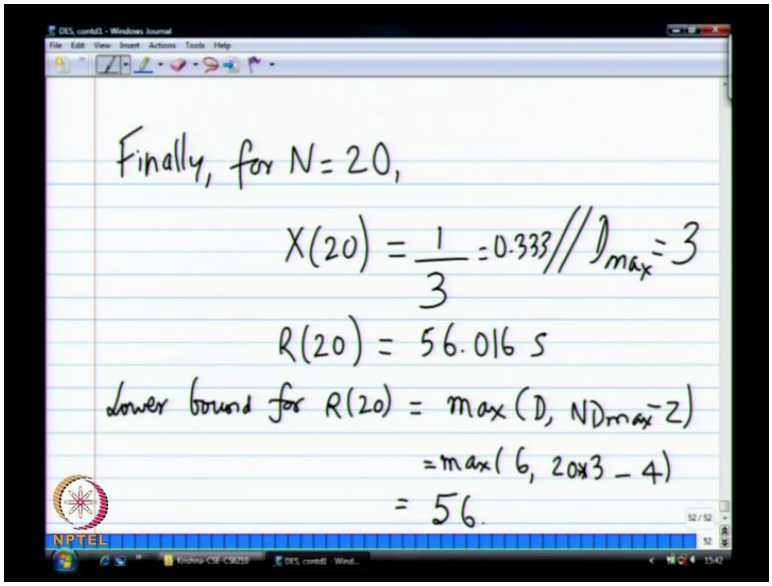
So, R I am going to just write, so the new delay is 7.4, because again if you (( )), so R cpu is now S cpu into 1 plus the most recently computed Q cpu, which is essentially 0.125 1 plus 0.2 right. That is what we computed that is equal to 0.15 and so on. So, we compute this way

right R A, R B. R A is 0.39 and so and so, all of these basically (( )) R. So, now the response time where one more job is increased to 7.4 seconds slightly higher than (( )), but still bearable one let say.

So, then X equals 2 by 7.4 plus 4, so now a throughput started increasing. And I am able to handle not getting two jobs per second, which is better than what I had before right; and then you can now compute the new Q cpu values. So, now your queues (( )) starts increasing, so this is 0.421; Q A is 0.684 like we did it last time right; X into V i into the last computed R i.

So, remember that R i is keep getting updated in every step, the Q i is also keep getting updated in every step. So, we will always use the most recent value for the (( )), V i does not change, but R changes Q changes and that is what is repeatedly used in this. So, dot dot dot that is the program just run this forever and we said N equals 20 right. So, for N equals 20, what should be the throughput finally?

(Refer Slide Time: 57:04)



So, what could be the throughput, N is a fairly larger now well not very large. But, what will be the upper limit for throughput? We will be getting the 1 hour D max limit right. And what is D max D max is, what is the value of D max? D A was 3, D B was 1 and D cpu was 2, so D max is 1 right. So, it should be 1 by 3, it cannot finish any better than 1 job every 3 seconds. So, from 0.1 I went up to 0.33 that is about the best I can do with this system. And then R of 20 equals again, so these is actually computed by your, you will find that this is ends up with

0.33 if you run it long enough. And R of 20 is again computed by our algorithm, which is coming out to 56 seconds.

So, the response time has gonna from 6 second to 56 seconds that is fairly you know depends on whether you want operate at that level or not, but that is what it is.

(( ))

The three demand values are 1 2 and 3, we never computed demand value for A

(( ))

V A into S i, S i is 0.3 and V A, 0.3 my 5 looks S looks like 5, so 0.3; so, R of 20 is 56 seconds. Now, what is the corresponding remember that for a (( )) making you get the lower bound right. So, from our bound calculations what is the lower bound right? This was again max D clearly D is 6 that is no issue. So, this will be max, so that will not sure, N is 20, D max was 3 and always think something write something Z, minus 4 right (Refer Slide Time: 59:56).

So, lower bound is 56 and our actual computed value is 56, so it is reasonable bound right. So, if we do not want to really go and figure this entire calculation, I can just look at this bounds and say look it cannot even have better than 1 by 3 throughput wise and you cannot do better than 56 seconds, you cannot go lower than that. So, these are just check, when you are do this in your exam.

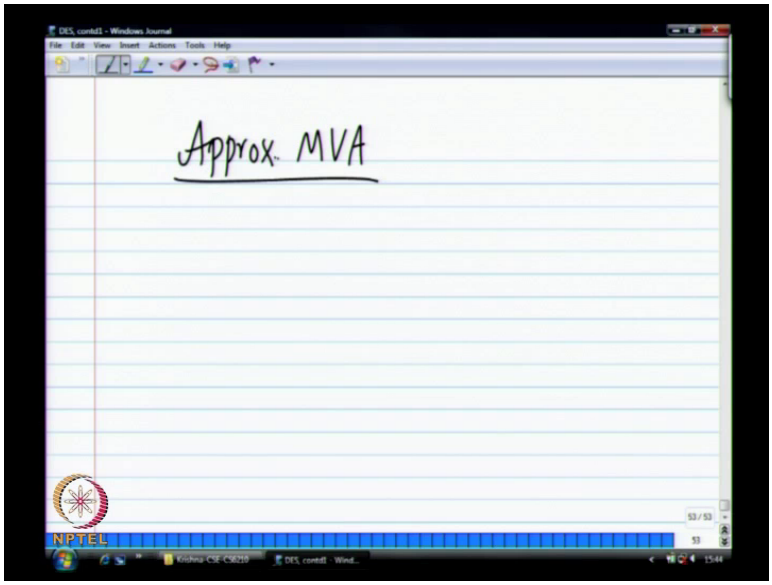
(( ))

That is after running this on the computer (( )) giving you the values right. So, you write (( )) a short program right, this is the simple program right. So, time of computations for the order N right, because each step you are not doing, you are just linear calculations is nothing.

So, order N is the computation time it is not bad by most standards, but people found that when N is very large let us say I wanted to do a million (( )) customer system you are looking at a web server transactions million or 10 million customers per second; you can be a (( )) running for 10 million customers to get the result right. So, (( )) is good, but the N complexity can be a deterrent, if your N is very, very large. So, that is what we always want to try to not

like 10 jobs per second, we are looking at million transactions coming to a city bank server we want to see what is the response time right.

(Refer Slide Time: 1:01:41)



So, to get around this order N business that have been lots of approximations, so there is one particular approximation that this book discusses which I want to present now. I am not sure this is still the best approximation. This is as 20 years ago, the books at this is the good approximation from like 1976 around that time right; some 70's 80's time frame that said, this is an approximated MVA; and that reduces your search time or a computation time from order N to some small order k, so an iterative algorithm, it will converge, it might not converge, but it gives you usually results in less than N time. So, if I give a very large N, you can simply iterate with this approximation and get similar values as what you would get with you regular MVA. So, we will come and look at approximate MVA on Friday right let see what the changes.