

Real - Time Systems
Prof. Dr. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

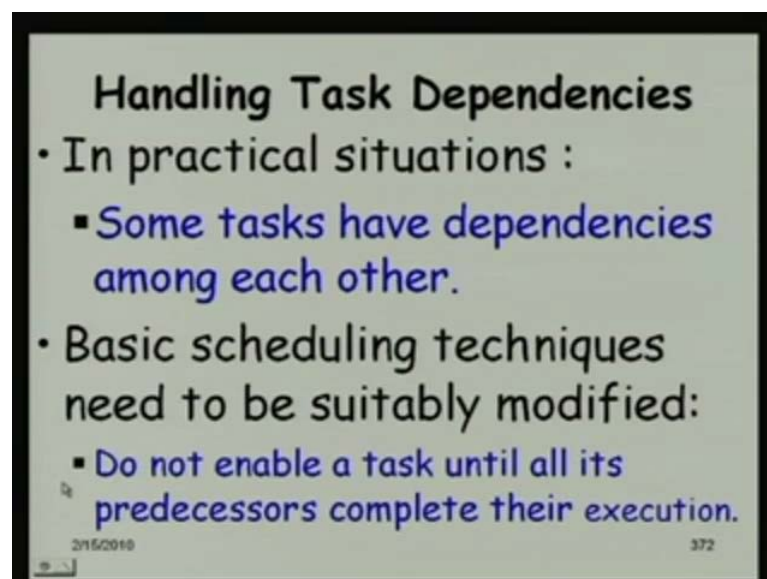
Lecture No. # 17

Real -Time Task Scheduling on Multiprocessors and Distributed Systems

Good morning, so, we will get started from where we had left last time; so, today we will first complete our discussions on what we were doing last class, and then, we will start discussing about how to schedule tasks in multiprocessors and distributed systems. So, these are some of the practical issues that we are slowly trying bring in; first, we had looked at the simple and ideal case of a uniprocessor, task scheduling on a uniprocessor, and then we were trying to bring in task dependences, and we saw that it is not really very difficult to handle task dependencies.

Today, we will complete our discussion on that, and then we will start discussing about, how to schedule tasks on multiprocessors in distributed systems, because nowadays, uniprocessors are slowly going out, being replaced by multiprocessors, and also distributed systems are becoming common place. So, with that let us proceed.

(Refer Slide Time: 01:29)



Handling Task Dependencies

- In practical situations :
 - Some tasks have dependencies among each other.
- Basic scheduling techniques need to be suitably modified:
 - Do not enable a task until all its predecessors complete their execution.

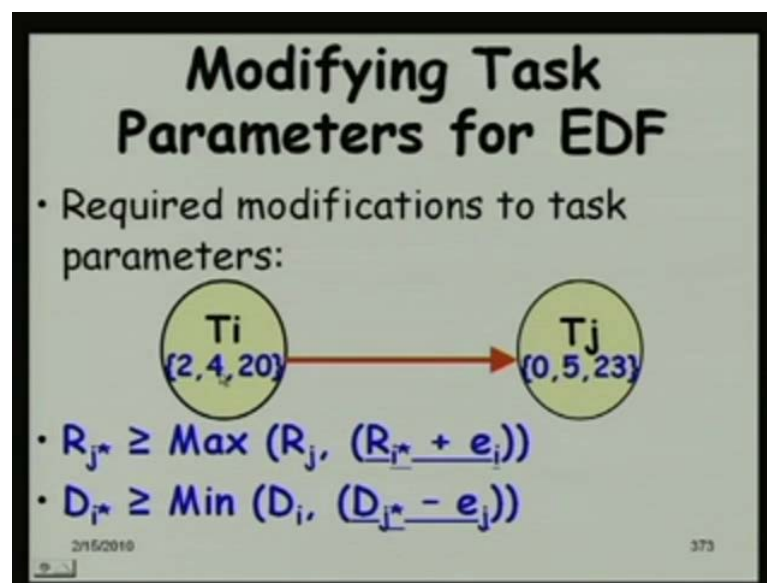
2/15/2010 372

So, first let us recall, what we are trying to do yesterday; we had said that in practical situations, some tasks have dependencies with each other, and that is what we had not assumed earlier, we had assume that the tasks are independent.

And then, we were saying that the basic scheduling techniques, the table driven, then the RMA and the EDF need to be modified, when there are dependencies among tasks, but the idea of handling task dependencies was very simple.

The idea was that do not enable a task, until its predecessors complete their execution. So, when there are dependencies, first let the task which have no predecessors complete, and then the tasks that are dependent on this become ready, because if they become ready before the predecessors are complete, this will compete for resources with the tasks, on which they are dependent.

(Refer Slide Time: 02:52)



We had looked at, how to modify task parameters, for table driven schedulers and also the RMA; basically, we are trying to modify the ready time, and we have taken some examples is, we found that it is not really very difficult; it is easy actually. Unless some of you, they want more examples on that, but looks like that its simple and you do not need additional examples; concepts are simple, on modifying task parameters for RMA.

Now, let us try to see, how task parameters can be modified for EDF. Basically, we will have to modify the ready time, so that a task does not become ready, until the

predecessor completes, and the predecessor completes, when its own ready time plus its execution time is over **right**.

Just see this parameter, this is for every predecessor for a task, we will have to find out its ready time, and its execution time, and check with the ready time, this we can say the phase actually, I could have even written here phase; the phase of task R_j , so the modified ready time per R_j is denoted by R_j^* . It should be equal to or greater than the maximum of these two parameters, one is its phase of the task j , and the other is all its predecessors T_i , its ready time - modified ready time and the execution time. So, until the task completes, we should not get the successor ready that is the idea here, and then since, the tasks are scheduled based on their deadline, we will also have to do modifications to the deadline of the tasks; this we did not have in the case of RMA. So here, obviously, we cannot increase the deadline. We can only reduce the deadline **right**, if a task completes before its deadline, it is alright; but we cannot increase its deadline. Now, first of all, why do we need to change the deadline of the task? Cannot we just change the ready time, and let it execute?

Deadline cannot be change, because the other task may be dependent and this may be critical task deadline anyway, this is the property of real time system.

No, no the deadline can be reduced. See, what you are saying is deadline cannot be changed.

Cannot be...

It is not **it is not** correct, because deadline can be reduced, cannot be increased.

But even if it is reduced, it will not going to solve an hyperbola.

It will have... It will be useful to change the deadline, because here the tasks are taken off for scheduling, based on the deadline **right**, the earliest deadline. In the RMA case, we change the task priorities. So, even among the ready tasks, we distinguished by priorities, but here, we will change the deadline, we will just take some examples, and we will say that we need to reduce deadline in some cases, where the previous tasks deadline minus execution time is less than this deadline **right**, then this will get a higher

priority a task, D_j will have a higher priority, compared to its predecessors, which we do not want **right**, that is the idea behind this second parameter.

So, let us take an example, of this one here; if we have a task T_i , whose phase is 2, the execution time is 4, and deadline is 20; and a task T_j , whose phase is 0, execution time is 5, and deadline is 23. Now, for T_j , let us compute the ready time; the ready time will be, maximum of R_j , which is 0 plus R_i star plus e_i . Since, it is the task, which does not depend on anything, R_i is same as R_i star.

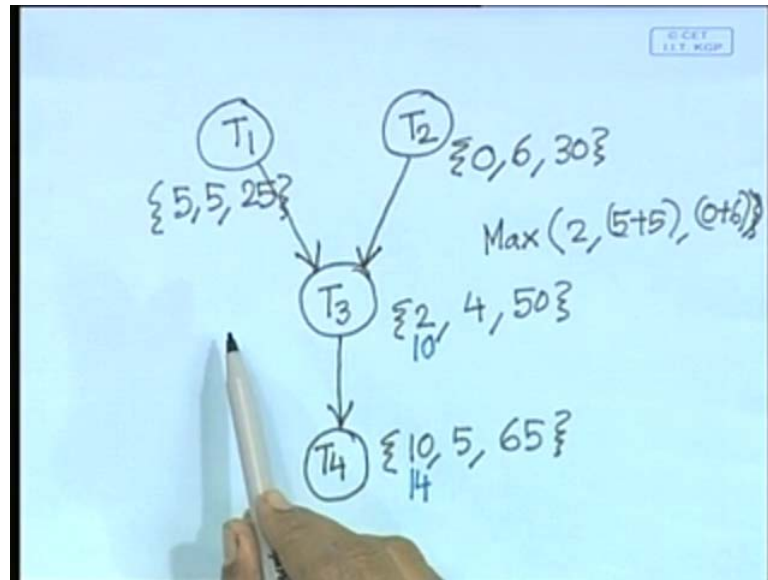
So, this is 2, R_i is 2, and e_i is 4, so this becomes 6 **right**. R_j is 0 and this is 6. So, we have to replace 0 with 6 here. The modified ready time for T_j should be 6. Now, let us say this is 23 is the deadline. Now, 23 minus 5 is 18 **right**; 23 minus 5 is 18. So, it must start by 18, otherwise it will miss its deadline, is it not?

It must start by it.

Yeah by 18, it must start by 18; otherwise, it will miss its deadline. So, for this task, we have to get the deadline as 18, because if it completes by 20, then there will be a problem here. **Right** this will miss its deadline, if it completes by 20. So, we will have to replace its deadline, task T_i with 18; **right** and with T_j 's ready time has to become 6. So, that is the modification we will have to do for EDF.

Now, let us just try some examples. I hope, this is understood everybody. Now, I will just give an example, please tell me, what will be the modified task parameters? So, let us just draw an arbitrary task graph.

(Refer Slide Time: 09:34)



So, let me just **a task**, draw a task graph like this; T_1 , T_2 . So, let us assume T_1 's phase is 5, its execution time is 5, and its deadline is 25. So, those are the task parameters for T_1 ; for T_2 , let us say it is 0 is the phase, and execution time is 6, and deadline is 30; and the task T_3 is dependent on both of this, and T_3 's parameters are, phase is 2, execution time is 4, and the deadline is let us say 50; and let us say, we have another task T_4 whose phase is let us say 10, execution time is 5, and the deadline is, let us say 65.

Just any arbitrary graph - task graph that I have drawn; now let us see, whether you have understood, how to modify the task parameters for EDF? Now, first let us look at the ready times. So, what about task T_1 and T_2 ? Do we need to change their ready times? No, their phase itself is the ready time. Now, what about T_3 ? What will be its modified ready time?

5, sir.

What about...why **why** the others are not speaking? What will be the ready time for task T_3 ?

10.

It will be 10, because that will be a maximum of 2, and 5 plus 5, and 0 plus 6, which is equal to 10. So, let me just write in a different color. This will be modified to 10. Now, what about task T 4? What will be its ready time?

14.

Yeah. So, the modified ready time for task T 4 will be 14, because that will be maximum of 10, and 10 plus 4, 14; is that ok? So, let me write here 14. Now, that is about the modification to the ready times, but what about the deadlines?

We start from the bottom, because T 4 must meet its deadline **right**, that has the longest deadline 65, and then for it to complete, we must at least 34 by 60; is it not? So, what should be the modified deadline for T 3?

Not, any minimum of 50 comma 60.

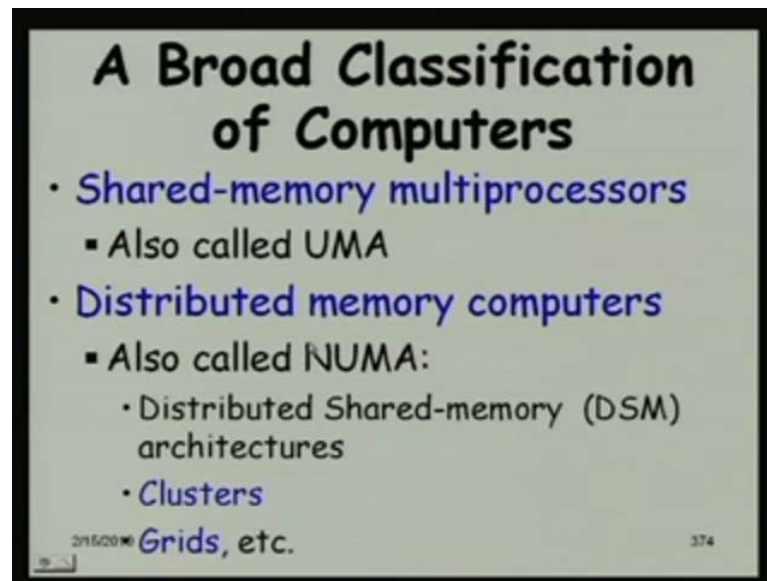
So, that will be minimum of 50 comma 60; which is 50 itself. So, nothing needs to change here, but what about T 2? Do we need change its deadline?

30 comma 40.

So again, there is no change to modify its deadline, because this is 30 comma 50 minus 4 is 46, 30 comma minimum of 30 and 46 which is 30, do not need to change its deadline. What about this one? This also, we do not need to change the deadline, which will minimum of 25 and 46 **right**. So, deadlines need not be changed, just by changing the ready times, we are able ensure that this precedence, adoring among tasks will be ensure during EDF execution. So, I hope that given arbitrary task graph will be able identify, what are the changes to the task parameters for it to execute according to the precedence relation, is that ok?

So, everybody says yes. So, we will not spend more time on this. Now, let us proceed from here.

(Refer Slide Time: 14:10)



So now, we look at scheduling on multiprocessors and distributed systems; because, as I was saying that these are becoming very popular, even the computers in our lab, laboratory are multiprocessors, dual core and distributed systems are becoming more and more common. So, let us first look at some of the characteristics of these systems, some basic ideas about this, and then will proceed from that point.

So, if we look at a broad classification of parallel computers, because as I was saying that even though so far, we studied uniprocessors, and they were quite simple, straight forward compared to parallel computers, but slowly uniprocessors are getting replaced with the multiprocessors and distributed systems.

So, let us look at some of the characteristics of this, so there are mainly 2 types: one is the shared memory multiprocessors - shared memory computers, which we will call as multiprocessors, and distributed memory computers, which we will call as distributed systems. So, the shared memory multiprocessors are also called as uniform memory access computers. Why is that? What is the implication, I mean, why are this called as uniform memory access computers?

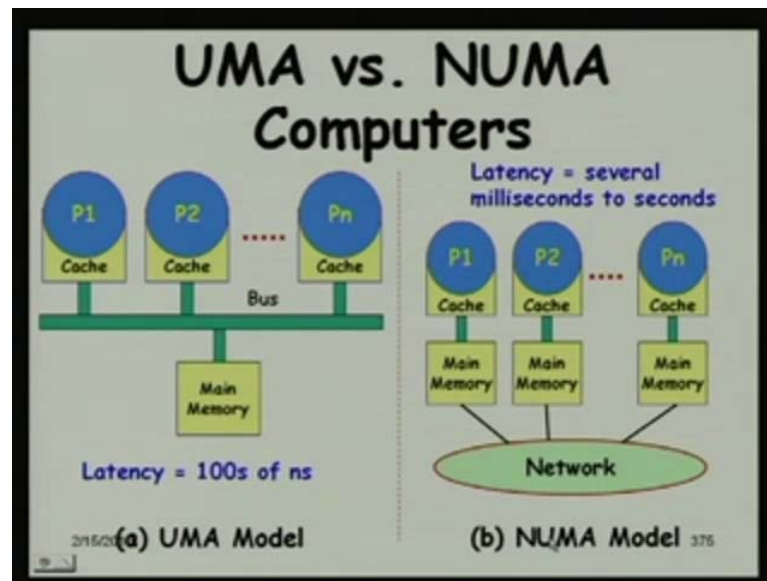
It is time to access the memory, primary memory **primary memory**, itself compared with the primary memory. So, that all the processors will be used conceptually.

Yes, that is the idea here. The memory is shared see here, the main memory is shared, and since it is shared, the time to access the memory that is for read and write operations, all processors take same time, similar time. So, this is uniform memory access; the time for every processor is the same, to access the memory, because that is a shared.

On the other hand, the distributed memory computers are called as non uniform memory access computers, because these are implemented, each computer has stores part of the memory. See, if the memory is distributed, see here, the memory is distributed among different processors, and therefore, the access time to the memory depends on whether it is local main memory, you are talking of or it is the memory that is present with some other processor that you are talking of. So, the time can vary widely from few milliseconds to few seconds, depending on where the memory is located.

And there are variations of this, distributed shared memory architectures. So, a distributed shared memory, this is done for simplification of programming basically. Here, the memory is distributed, but the operating system makes it appear to the application program that it is a shared memory. Just a operating system facility for programmers, because programming shared memory computers is easy, just keep on writing, just like you used to do for uniprocessors, and the the operating system takes care of where the memory is located, and generating appropriate commands to do the read and write operations, and we have the clusters and grids etcetera, which are becoming popular.

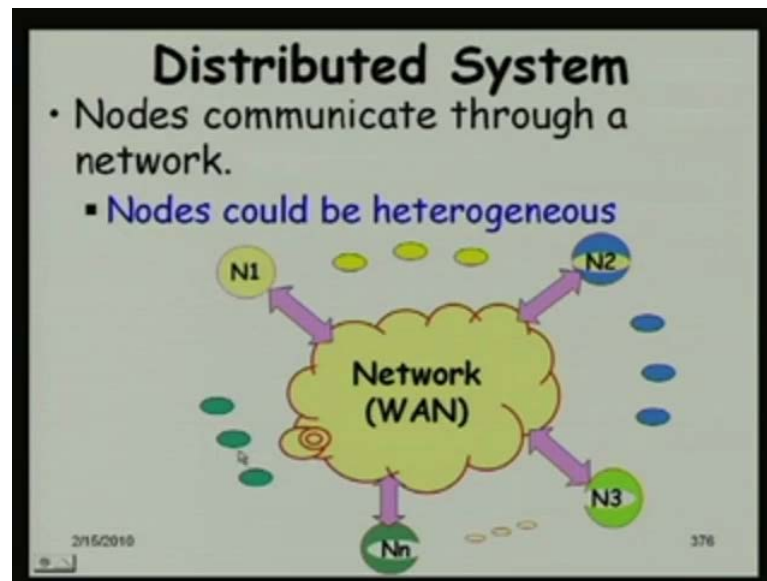
(Refer Slide time: 18:02)



So, this is the schematic diagram of a uniform memory access model, which is a multiprocessor. So, we have the different processors have their own cache. And the main memory is shared. So, that is what, we mean by the shared memory, the main memory is shared and they are interconnected by a high speed bus.

On the other hand, in a distributed system, we have a non uniform memory access model, where the different processors have their own cache and main memory; and they are interconnected by a network. So, a processor needing data from its own memory will be very fast; whereas, if it needs data from another main memory, it depends not only on the access time for this main memory, but also the network delay **right**. So, the latency here is several milliseconds to seconds, depending on what, where the memory is located, whereas, here it is about a millisecond or may be hundreds of nanosecond.

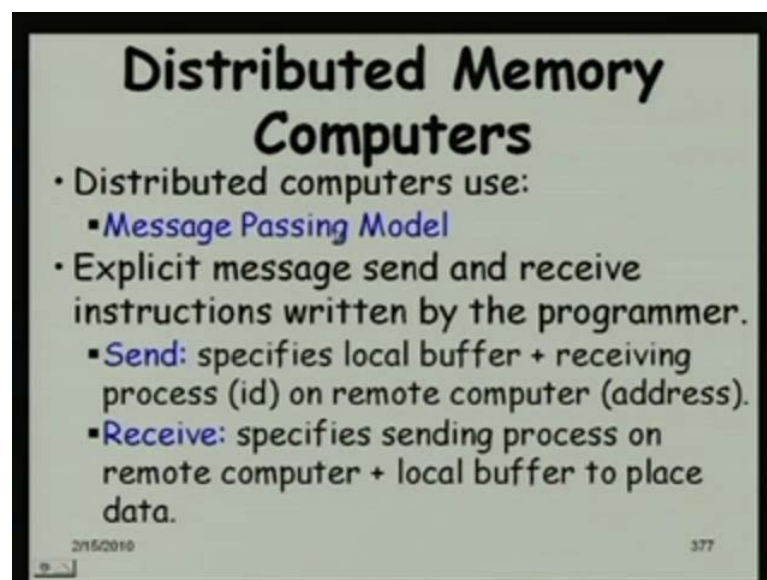
(Refer Slide Time: 19:22)



So, this is just a different schematic for a distributed system; the network, typically we talk of a wide area network, but it can be also a local area network. A distributed system can also be implemented on a local area network, where we have different nodes, which typically are heterogeneous nodes.

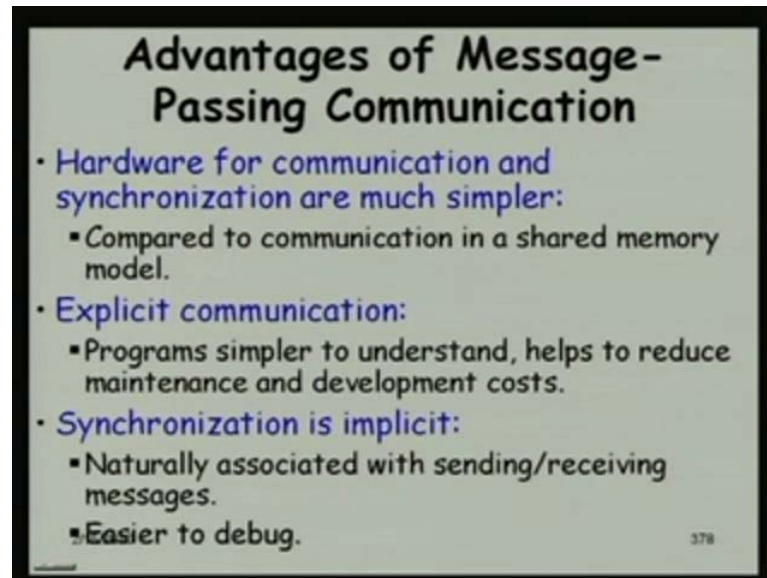
So, that is why we have shown them in different colors. Some may be desktops with windows may be... Some may be linux computers etcetera.

(Refer Slide Time: 20:07)



The distributed computers use a message passing model, and here the programmer has to write explicit, send and receive instructions **right**, whenever the data is located on a different processor, needs to write receive and send command.

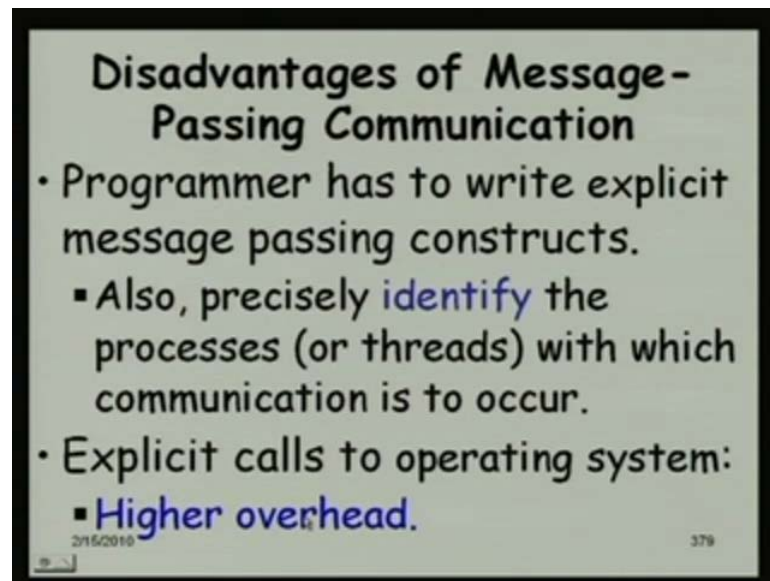
(Refer Slide Time: 20:38)



And the advantages of message passing communications are many compared to a shared memory, even the shared memory we said that its very simple to program, but message passing communication has its advantages: One is that, the communication and synchronization is simpler; actually during every message passing synchronization takes place. Is it not? Because unless the receiver is ready, a sender cannot send; so, implicitly there is synchronization, during every message passing primitive.

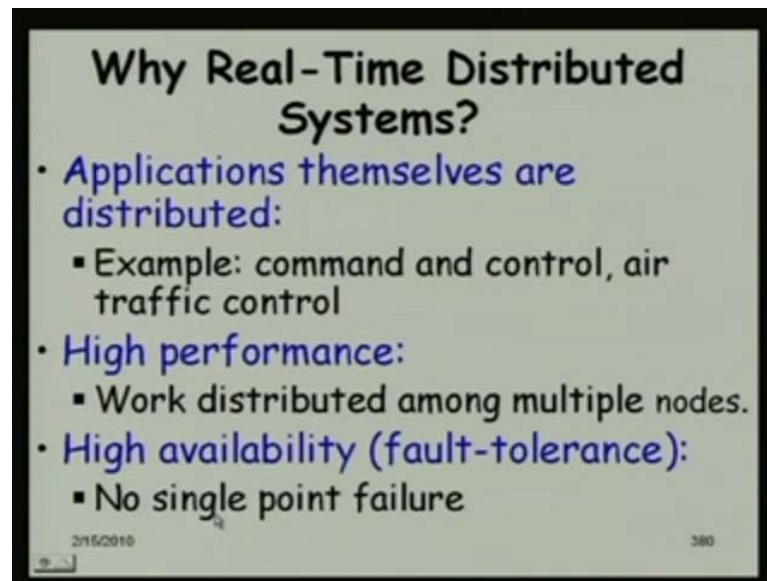
Whereas in a shared memory it does not matter, whether the other process is ready or not? It just modifies it. Here, the message passing makes the programs easy to understand, because you know, at what instant communication occurs, with which process etcetera, but in a shared memory computer, it just modifies a process, modifies its data, and it may be any process which accesses it, is not clear actually; just by looking at the program, it is very difficult to deduce that after a change occurs, which other process will access it, read it and modify it, which will read first etcetera, but here its explicit, it will name the process, to which the data will be sent. It is there explicit, the synchronization is implicit, easier to debug.

(Refer Slide Time: 22:17)



But there are also disadvantages with the message passing communication; that is why we find that multiprocessors are possibly used more heavily than distributed systems. One is programmer does extra job, writes message passing constructs; you have to precisely identify the process to which communication is to occur, and when there are dynamic creation of processors etcetera. You will have to keep track of the process ids, and then communicate with them, and also each time there is a message passing; you will have to give explicit calls to operating system, which is a higher over head. So, that way, you can argue that shared memory programs are more efficient compare to message passing programs.

(Refer Slide Time: 23:17)

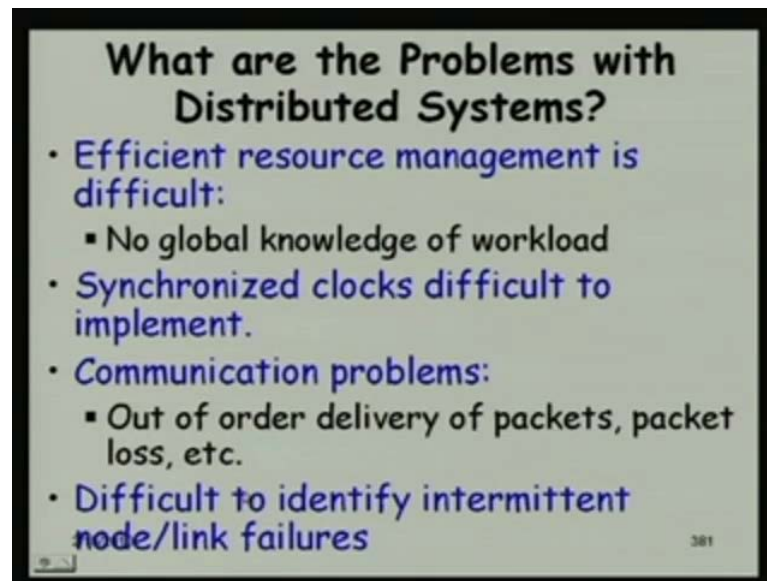


But let us see why the distributed systems and multiprocessors they are becoming more and more important in real time applications? First, let us look at the distributed systems; in many applications, the application itself is distributed. For example, there might be sensor collecting data from different locations **right**, physically separate locations, and you cannot really transfer all the data to a central processor, and then execute it. What really happens is that these are processed locally near the sensors, and then a summary data is sent to the computer, and even there, it need not be just one processor which takes action.

It may be different processors, which are there, which handle different aspects of computation. Many examples are there: command and control, air traffic control etcetera; another reason, why distributed systems are becoming popular is their higher performance, because the work load by definition is distributed among multiple nodes.

And also, fault tolerance can be easily implemented here, it can be different nodes and no one node fails, other nodes take up from that point; and whereas in a uniprocessor, if the processor fails, the system fails **right**; single point failure can occur.

(Refer Slide Time: 25:02)



But before we look at the distributed system, let us also understand, what are the problems with the distributed systems? One is resource management is difficult, because at anytime, we do not know, what are the workload of different processors? It is very difficult to keep track instantaneously, what is happening in different processors, because even if they exchange their information, but still by the time, it comes and reaches, it will become obsolete. And also you cannot just keep on transmitting, what is the load position at different nodes that will be too inefficient. So, we will have to... if at all transmit the load positions, once in a while **right**.

So, that means, you have a absolute information about load position and different nodes. So, once a task arrives, where it needs to be executed, is a very difficult question, because you have to work based on only absolute information. The systems information might be, there are some **load** node is relatively free, but if the task is that migrated to that node, find that that node has already got lot of a work, very recently.

So, that assumption may be wrong, because of the stale information. So, these are the problem, we will see how to address this. There is no satisfactory solution to this exist, it is a problem inherent to all distributed system. Global knowledge about the resource uses, and the load at different nodes, does not exist. The global information that is current or accurate does not exist, only at best, we can have some staled information **right**. So, that is one problem.

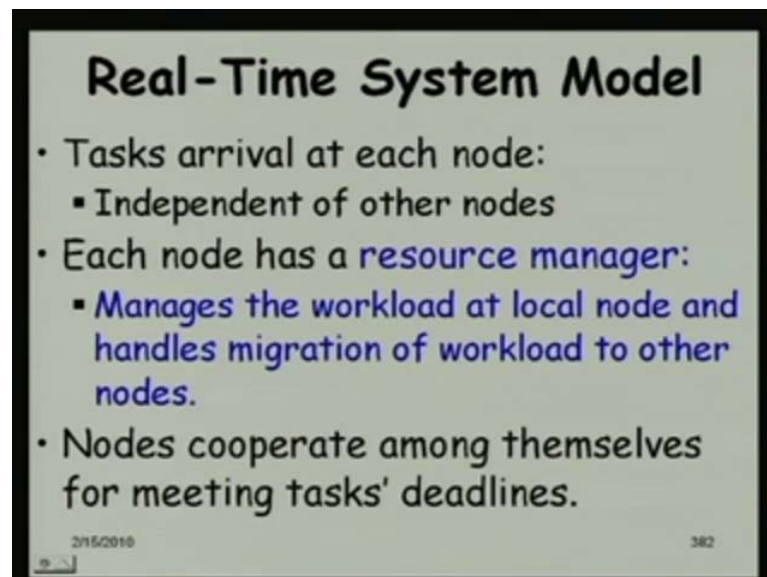
Another is that in a uniprocessor or in a multiprocessor, the clock is one, but here each node will have its own clock, and synchronizing them is difficult. We will **we will** see the problems in clock synchronization.

And then, also distributed system needs to address the communication problems. The packets might be delivered **out of that** out of order, there may be total loss of packet, not only packets are delayed out that, but they can also be loss. So, please corrective actions need to be taken and if the task is a real time task, then it makes it more complicated.

What to do when there is a packet loss? For example, and in a distributed system, it is very difficult to identify intermittent node failures, you do not know, whether it is due to a congestion that you were not able to communicate with a node or is it that it has actually failed intermittently **right**.

So, very difficult questions here, we have to address; and we will see that as a result, the scheduling solutions are also very difficult, compared to a uniprocessor.

(Refer Slide Time: 28:33)

A slide titled "Real-Time System Model" with a black border. It contains three bullet points. The first bullet point is "Tasks arrival at each node:" followed by a sub-bullet "Independent of other nodes". The second bullet point is "Each node has a resource manager:" followed by a sub-bullet "Manages the workload at local node and handles migration of workload to other nodes." in blue text. The third bullet point is "Nodes cooperate among themselves for meeting tasks' deadlines." At the bottom left, there is a small icon and the date "2/15/2016". At the bottom right, the number "382" is displayed.

Real-Time System Model

- Tasks arrival at each node:
 - Independent of other nodes
- Each node has a resource manager:
 - Manages the workload at local node and handles migration of workload to other nodes.
- Nodes cooperate among themselves for meeting tasks' deadlines.

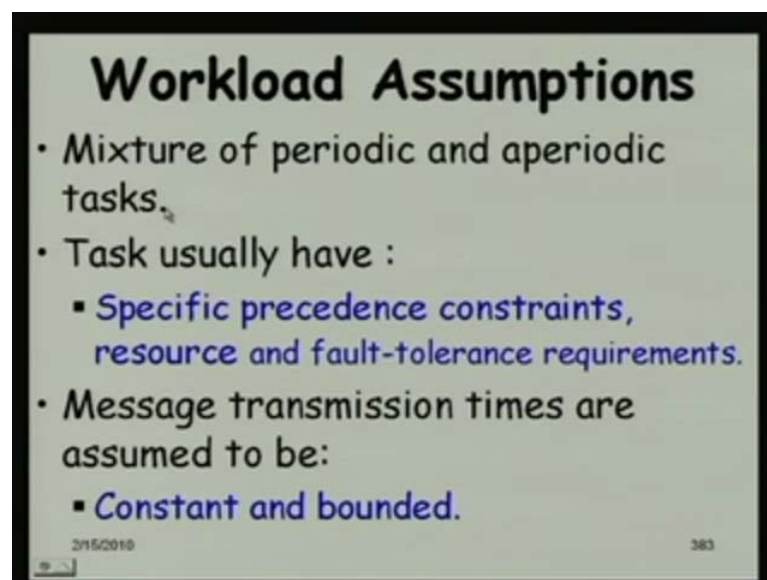
2/15/2016 382

Let us proceed, before we look at the scheduling... Let us look at the system model. The tasks arrive at each node independent of the other nodes. So that means, the sensors for example, are distributed, they collect different data or may be same data from different locations, and the rate that at which the tasks get generated at that node is independent of

the other nodes, because they get different types of data at different rates, and from different locations.

And typically each node, we will assume to have... We will assume that it has a resource manager. The responsibility of this resource manager, we have to manage the workload at the local node, when the tasks arrive, it will try to check, whether it can be processed locally and if not this resource manager would have to find out, where to transfer this task? That is migration problem, and we will also assume that the nodes will cooperate among themselves, they might exchange messages. Other information, they might honor requests from other nodes for migration etcetera, to meet task deadlines. So, that is the basic assumption, we will make as far as the system working is concern.

(Refer Slide Time: 30:06)

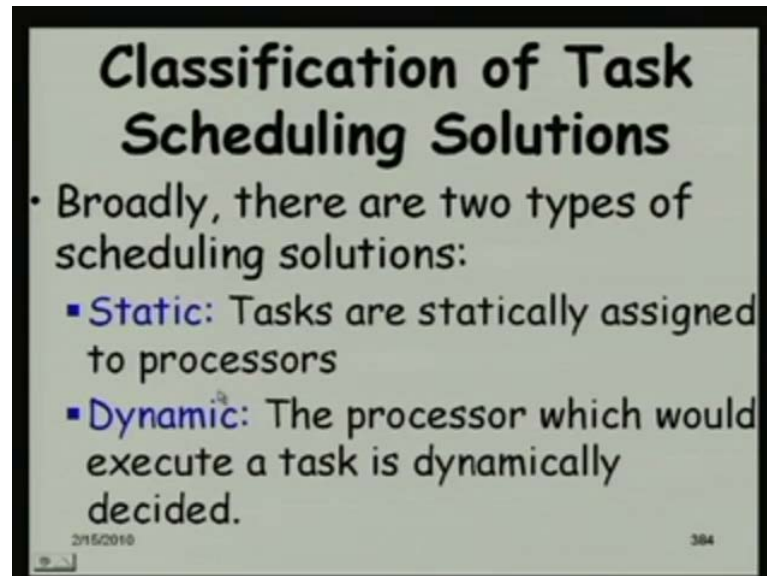


We will also make some assumptions regarding a workload; the workload will be a mixture of periodic and aperiodic tasks. The tasks can have precedence constraints, specific resource requirements, and also a specific fault tolerance requirements, and for simplicity. We will assume that the message transmission times are constant, and bounded.

We know that, this is not true actually, because, messages depending on the load condition, congestion, they might take very different times in a wide area network, but we will, right now, assume that we have some protocols, which will ensure that

transmission times are constant and bounded; towards the later part of this course, we will see how this can be achieved; we will not be able to achieve 100 percent a constant and bounded; but for specific tasks, we will call this as the quality of surface for a communication and we will see that constant and bounded quality of service. We will try to ensure through use of proper suitable protocols.

(Refer Slide Time: 31:28)



So, let us look at a broad classification of the task scheduling solutions; we have broadly two categories of solutions: One is a static solution; in a static solution, we predetermine, which task will run on what processor and using what algorithm; the static solutions, the tasks are statically assign to the processors; on the other hand, we can have we will discuss another category of solutions, which is the dynamic solution, where once the tasks arise, it will be each time decided where the task will execute, will it execute on local node, will it execute on node x or node y that will be decided on a task to task basis; of course, the static schedulers are simpler, is done just once, but the static schedulers, they cannot really handle when the task themselves are not static. See, the tasks are nice periodic task, and we know, where they are coming etcetera.

We can use the static solution, but what if, we do not know when the tasks will arise, and how many of them will arise etcetera, will we do not know, we cannot really use a static solution, is it not?

(Refer Slide Time: 31:28)

Classification of Task Scheduling Solutions

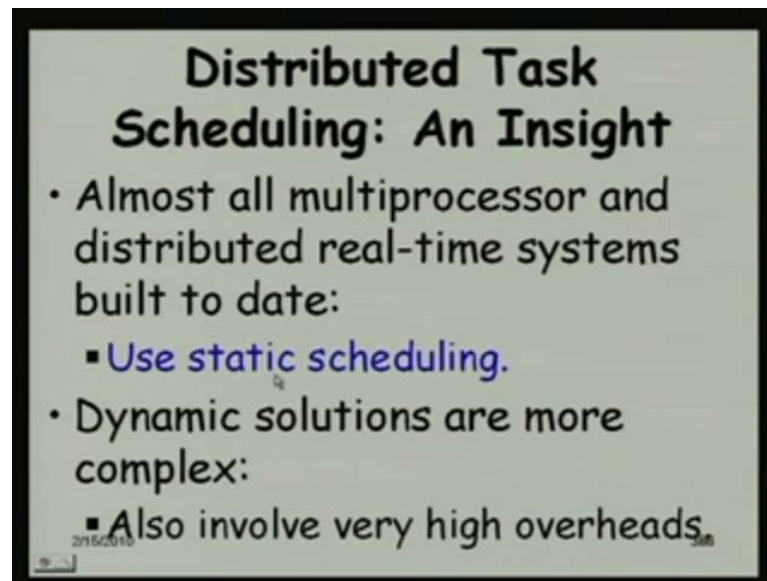
- **Local scheduling:**
 - A resource manager is associated with each node.
 - Task scheduling, and load transfer.
- **Global scheduling:**
 - A single scheduler handles balancing load across nodes.
 - Transfer policy

2/16/2018 385

We can also have another classification that is local scheduling versus global scheduling. In a local scheduling, the resource manager will **will** be associated with every node, and which we will look at whether the task can be executed locally or need to be transferred to another node etcetera. So, that is for the every node, the decision will be separate based on a local scheduler, **sorry** a resource manager that is running on the node, every node have a resource manager. On the other hand in a global scheduling, we will have just one node decide about what to do? Once when task arises, this is the one which will run somewhere and it will decide, what to do with that task, where to send.

So, obviously, what is assumed here is that the global scheduler has knowledge about what is the resource position, at resource utilization position at different nodes, and we had said that it is very difficult to have global state information. **And as result...** These global schedulers have their limitations, and the ones that we will discuss. The solutions are local scheduling; global scheduling has its disadvantages.

(Refer Slide Time: 34:26)



Now, let us look at what type of schedulers to use, whether to static or dynamic. It is not surprising, it would not be surprising to know that all multiprocessors and distributed systems that are commercially implemented, use a static scheduler, because these are easier to implement. You can ensure time constraint and so on. The dynamic solutions are more complex, higher overhead, but that is the situation right now, we do not know, about may be few years from now, we can have the dynamic schedulers also being used heavily.

Because, the tasks are becoming more and more complex task, characteristics are becoming complex, which we may not be able to handle with static schedulers. So, even if these are high overhead. Let me, just ask this question that why should the dynamic solutions have high overheads? Compared to a static scheduler, why should dynamic schedulers have high overhead? We have written here; they involve higher overhead compare to static schedulers. What **what** do you think? Why do these have higher overhead?

(Audio not clear. Refer Time: 35:54 to 36:00)

Any, any other...

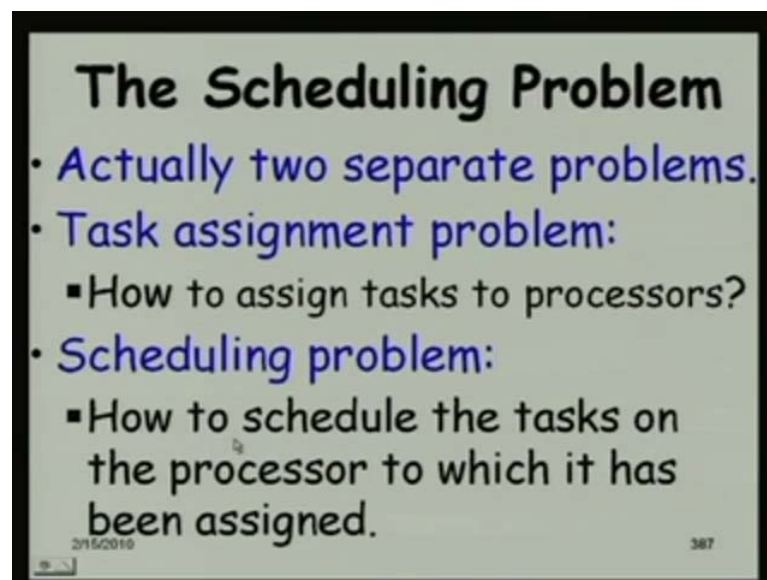
Because dynamic solution will involve high information exchange and repeated computation.

Any other answer?

There was context switching and...

Not really, see the thing is that see, static one is just done once; is it not? Just done and whereas, a dynamic one, we need to have several types of information exchange, the system state has to be with there with every resource manager locally, and then as the task arrives, decision has to be done **right**, and the node the task might have to be migrated and so on. It is has to done for every task where as in static one, it is just done once for well. So, that is the main reason.

(Refer Slide Time: 36:49)



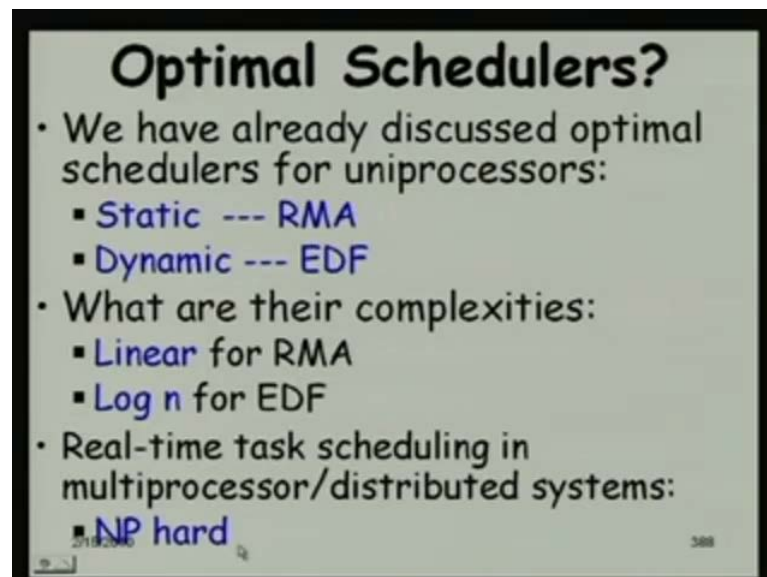
Let us proceed; in this multiprocessor and distributed systems, the scheduling problem is actually two separate problems, for uniprocessors we had see that the scheduling is only, how when to schedule the task? When the task will start?

But here in all multiprocessor and distributed system, we will not only have to decide when a task starts, but before that we have to decide which node or which processor the task will run; that is the first thing, and then once it has been assigned, we need to worry about when it will start on that node.

So, the assignment to a processor is the first problem that we need to worry about, we will call it as the task assignment problem, which will basically deal with, how once a

task arises, which node it should be migrated to should it be run locally at some node or should it be migrated to some node, that the task assignment problem; and then is the scheduling problem; once the assignment is done, we know that which node it is going to run or which processor it is going to run, then we worry about when it will run.

(Refer Slide Time: 38:13)



Optimal Schedulers?

- We have already discussed optimal schedulers for uniprocessors:
 - Static --- RMA
 - Dynamic --- EDF
- What are their complexities:
 - Linear for RMA
 - Log n for EDF
- Real-time task scheduling in multiprocessor/distributed systems:
 - NP hard

But what about optimal schedulers for multiprocessors and distributed systems, please, recall it that when we were discussing about uniprocessors. We were actually saying that see, there are large number of algorithms proposed, and we said that there are optimal scheduling solutions exist, and if we know that we know, more or less about that area, and for static schedule static task scheduling - static priority scheduling, which was the optimal?

RMA.

Yeah, Rate Monotonic Algorithm was the optimal. We had said that, if there are any scheduler can schedule a set of tasks successfully, then RMA can schedule, but whatever RMA can schedule, the other scheduling algorithms may not be able to schedule right. And similarly for dynamic priority, we had identified EDF, and we had studied these two algorithms and we said that, if you know this the RMA and EDF and also the deadline monotonic algorithm, because that is for different task characteristic that is optimal.

But unfortunately for distributed systems, we will see that no such optimal solutions exist. Let see the reason and what is the status here? If there are optimal schedulers existing here, we could have just studied 1 or 2 algorithms, and said that see, these are the optimal schedulers have to be used. And we will have to do that, but here no such clear got answer here. Let us **lets** see the situation here.

We had seen that for uniprocessors; as far as static priority schedulers discussed about RMA, the optimal scheduler, for dynamic priority EDF was the optimal scheduler. And then we studied about their complexities. What is the complexity of RMA by the way? I am saying, run time complexity - time complexity of RMA.

(No Audio. Refer Time: 40:27 to 40:34)

11.

So, what is the... you have to look at the algorithm, what **what** are the steps it does **right** and based on that you have to compute the run time complexity. Is it not? Time complexity, so what are the steps in RMA?

Maintain the necessary conditions (Audio Not Clear. 40:54 to 41:00)

So, once a **once a** task arises, what you need to do?

Scheduling time, we have to arrange the in order of priority.

So, the tasks are in order of their priority, is it? So, is that a queue or I mean how will that be implemented? But why a queue... We had discussed about the implementation of RMA, I hope somebody remembers that **how is the**...

(Audio Not Clear. Refer Time: 41:30 to 41:36)

Any, any other answers, please recall it, you possibly you have to revise little bit. See, we had said that every operating system has a set of discrete priorities **right**. And here, in a rate monotonic algorithm, it is not the deadline; like some of you were trying to say that see, you will have to examine the task you each time, no; its basically we have to scan through priority list. So, if there are 16 priorities, check whether priority one tasks exist, execute them. There are no priority one execute priority two.

Right and for every priority, we just order that way in which the tasks arrive; first in first out. **Right**, for a different priority levels, so what will be its complexity? There are only a finite number of priorities 16 or 8 or maximum 32, we will discuss about 32 priority systems; and then you will have to each time a task arises, you have to scan all this levels, The first one, you find task highest priority level execute it. So, what will be the complexity?

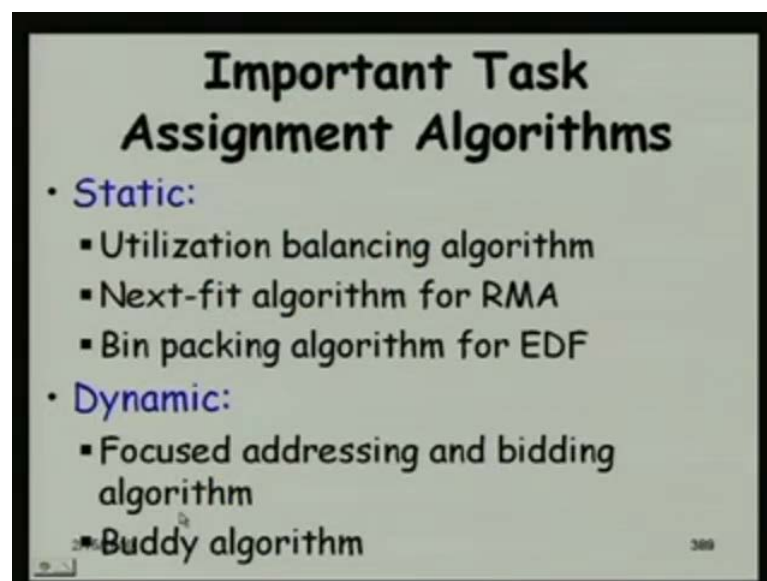
(Audio Not Clear. Refer Time: 42:53)

Yeah, it is a linear complexity.

Say, linear complexity for rate monotonic; and $\log n$ for EDF; it will be $n \log n$. I am sorry; it will be $n \log n$, because this will for every task **right** $n \log n$. I am sorry, I done a mistake here.

But unfortunately for real time task scheduling on multiprocessor and distributed system, this is a NP hard problem; it has been proved on papers have been published. If you are interested to show, how to show about real time task schedulers are NP hard. You can just search it on the Google, you will find enough papers where there so different type of systems, different types of task characteristics, and how they have been shown to be NP hard.

(Refer Slide Time: 43:49)



Important Task Assignment Algorithms

- **Static:**
 - Utilization balancing algorithm
 - Next-fit algorithm for RMA
 - Bin packing algorithm for EDF
- **Dynamic:**
 - Focused addressing and bidding algorithm
 - Buddy algorithm

389

Those who are more theoretical inclined, they can check for a proof for NP hardness, but right now, we will just assume that the tasks are NP hard, the schedulers are NP hard; a scheduling problem is NP hard.

Now, first look at the task assignment problem, because we had said that task scheduling in multiprocessor distributed systems. The task assignment is the first problem, we need to worry about and having done the task assignment, then we will worry about task scheduling problem. So, the task assignment problem, there are actually two types of solution: one is the static solution and the other is the dynamic solution.

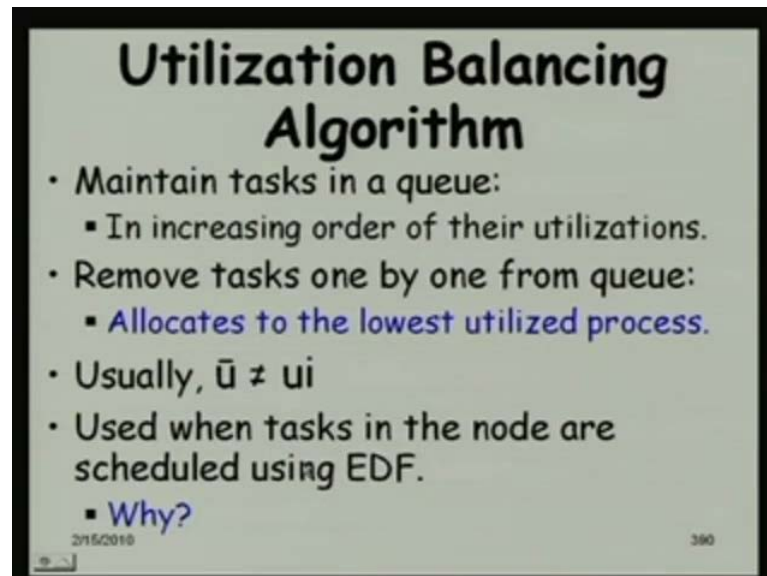
So, let us **lets** see these details about the solutions. We will discuss three algorithms actually, on the static task assignment solution. These are utilization balancing algorithm, next fit algorithm for RMA, and the bin packing algorithm for EDF. So, the next fit algorithm will be used.

When the nodes themselves are scheduled using RMA, and the bin packing algorithm will be used, when the nodes themselves. So, these are the scheduling algorithms to be used in the individual nodes, and this is the task assignment, which will be used in conjunction with this scheduler. We cannot use a bin packing algorithm for RMA, because it will not work; we will see why, and then we will look at the utilization balancing algorithm, which we will **we will** see that we can make it run with RMA and EDF; both we will **we will** just examine this algorithm. And then we look at two categories of dynamic solutions. The focused addressed and **bid** focus addressed and bidding algorithm and the body set algorithm, here these distributed task scheduling; task schedulers were distributed and multiprocessor system;

Thousands of algorithms have been proposed most based on heuristic. We will see that these are all heuristic, because the problem is NP hard, we will have heuristic solution, just try to based on some assumption, we just try to check how they are performing for specific systems and task characteristics, but optimal solutions do not exist, and the as you are saying that the uniprocessors, most of the results are available in the 1970s and some results, we had seen in 1980s also Liu and Livovsky's criterion and so on, but here, the research started actually in 1980s; and even now, you will find that many research papers are getting published here, PhD. degrees are awarded for real time task scheduling on multiprocessors and distributed systems, and now that we have the multi core

computers, which are becoming common place. So, that is again another hot area, for research in developing suitable task schedulers for real time tasks.

(Refer Slide Time: 47:35)



Utilization Balancing Algorithm

- Maintain tasks in a queue:
 - In increasing order of their utilizations.
- Remove tasks one by one from queue:
 - Allocates to the lowest utilized process.
- Usually, $\bar{u} \neq u_i$
- Used when tasks in the node are scheduled using EDF.
 - Why?

2/15/2018 390

First, let us look at the utilization balancing algorithm; the idea is very simple here, we maintain a queue; the queue will contain the tasks in increasing order of their utilization obviously, we assuming that their periodic tasks **right**, then we can compute the utilization.

Remove tasks one by one from the queue, so, the first task in this queue will be the one having lowest utilization, and the last task in the queue will be having the highest utilization, and how is the utilization defined - utilization for a task?

(Audio Not Clear Refer Time: 48:26)

Yeah, e_i by p_i execution time over which period; so, after p_i , the next instance of the task will occur. So, for during every p_i , we need e_i for to execute this task or in other words, it will keep the processor b g for e_i by p_i fraction of time **right**. So, that is the utilization.

So, remove the tasks one by one from the queue; first take out the task having the lowest utilization and then allocate it to the lowest utilized processors. So, we will also have the processor status information, if just take one task, then examine all the set of processors,

we might have 4 or 2 or something, and then check which is the process- the processor, I have written here process would have been processor, check which is the lowest processor at that instant, and then allocate it of course, for the first task from the queue both the processors will have 0 utilization. It be allocated to any one of them, but then the second one, naturally we will have to allocate to the one which has zero utilization, and then from third one, we will just compare, keep on comparing, the utilization for different processors, and then allocate that.

But unfortunately, in this algorithm, it is very difficult to achieve the loads or the utilization of the individual processors equal to the average utilization. See, if you have n tasks, then the average utilization of the processors will be $\sum U_i$ by n . Is it not? See if each task, I think, I also using the U_i symbol for the processor utilization. Let me just say that the utilization for a task is let us say, t_u .

So, the utilization due to all the tasks will be $\sum t_u$ **right**. So, if we have a perfectly balanced load, then we will have $\sum t_u$ by n should be the load for every processor, but unfortunately, here we will have a variation of load among different processors. Some load will be more loaded, some node will be some processor will be more loaded than other processors, and therefore, we are saying that \bar{U} , which is the average node load, will not be equal to the individual node that is actually the figure of merit for any balancing algorithm.

So here, if we do not restrict the utilization for the number of nodes, it can only be used for EDF. See, if we say that see, we go on assigning tasks, until there is certain utilization achieved, it can only be used for EDF. Why? Because... Anybody would like to answer, before I answer? Why is it that? If in this form of the algorithm, it can only be used for EDF.

Yes, anyone wants to answer this question? Answer is simple actually, see here, we are just trying to balance the utilization, but you know that in RMA, the schedulability criterion for real time task is that there is a bound on utilization, **depending** that is dependent on the number of tasks, and that **that** are allocated to the node and we have to looked at the liu layland and liu livosky's condition, and here in this utilization balancing, just check that we have not really followed any of those constraints of liu layland or liu livosky. So, this algorithm will not work for RMA in this form. Is it not?

We are just worried about, how to balance the load. So, it might happen that some node has 0.9 utilization and another node has point 8 utilization.

It will ensure that EDF, will they can be run on EDF, as long as the utilizations is less than 1, but for RMA this algorithm will not work in its current form. So, how do we need to change this algorithm to work for RMA? So, please think about it. How can the utilization balance in algorithm be modified such that it will work for RMA?

So, we will just take a short break for the next class, before we start about discussing other static scheduling solutions and also the dynamic scheduling solutions for distributed systems right. So, we will we will stop here.