Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

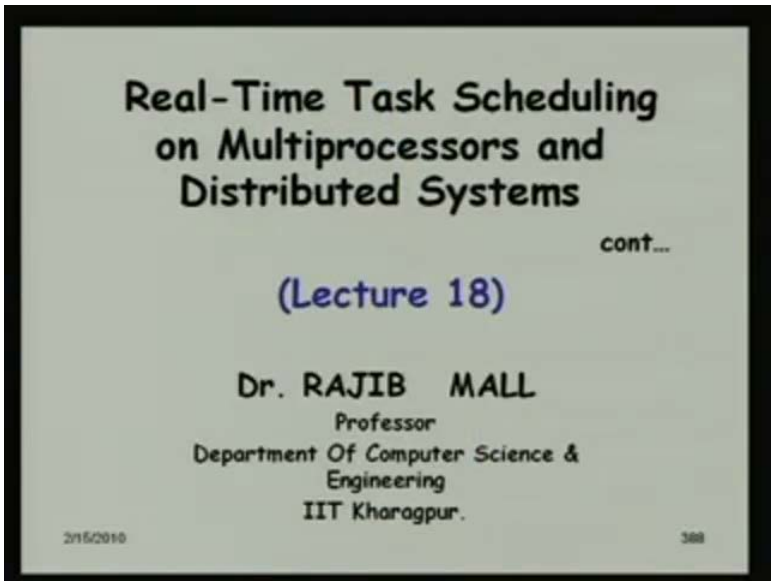
Module No. # 01

Lecture No. # 18

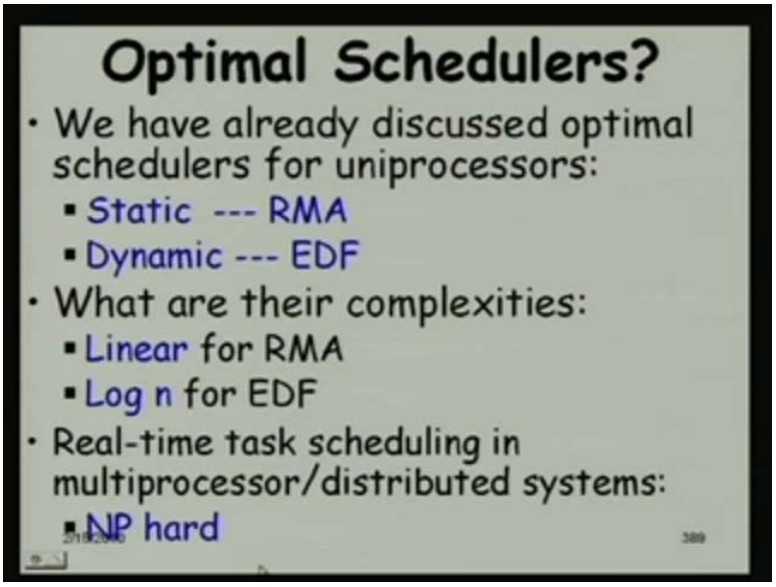
Real-Time Task Scheduling on Multiprocessors and Distributed Systems (Contd.)

So, let us get started and we will continue our discussion from the point we discussed last time. So, if you remember we are discussing about task scheduling in multiprocessors and distributed systems.

(Refer Slide Time: 00:38)



(Refer Slide Time: 00:40)



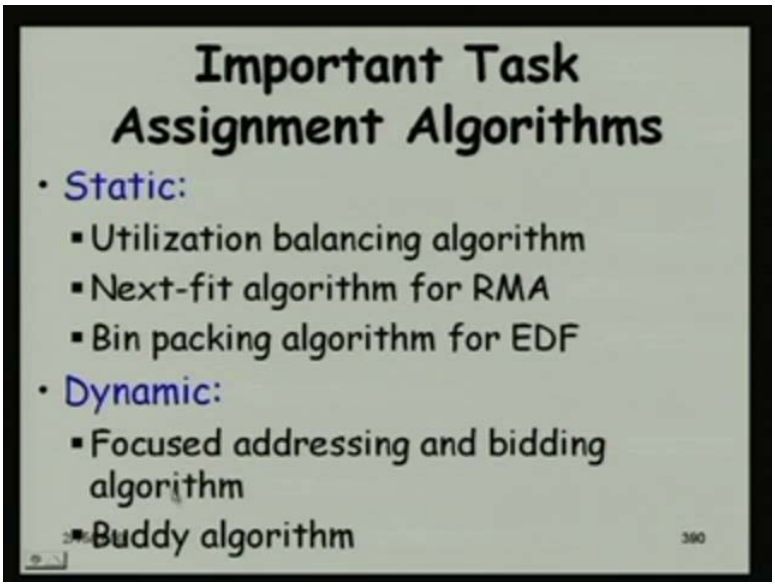
So, we had said that for any processors you could discuss just two algorithms, one was a optimal scheduler for static priority algorithm and for dynamic priority algorithm and that summarized our discussions on the uniprocessor system, but for real time scheduling on multiprocessor and distributed systems say n p hard problem and we have many heuristic solutions proposed our ears and some of them are popular.

And we are going to discuss only a few popular scheduling solutions and we had said that, there are two categories of task assignment algorithm the task assignment needs to be done before the tasks can be scheduled on the individual nodes.

And the assignment can be static or dynamic; in static we are going to discuss three algorithms. The utilization balancing algorithm which can be used for E D F and we can modify to use for R M A, the next-fit algorithm for R M A and the bin packing algorithm for E D F. These are three static solution just sample static algorithms we will discuss not by no means you are saying that these are the best algorithms that are available.

What we are saying at? That this three are popular algorithms many variations of these algorithms say just. And similarly, for dynamic there are many scheduling solutions that have been proposed we are discussing two of them the focused addressing and bidding algorithm and the buddy set algorithm.

(Refer Slide Time: 02:31)



In the utilization balancing algorithm we had seen that we need to maintain a task queue, which has the tasks in increasing order of their utilization. Remove the tasks one by one form queue. And allocated to the lowest utilized processor, each time you take a task out of the queue examine all the processors find the one which has the lowest utilization and just allocate it to that simple algorithm.

And here, the node utilizations are not really perfectly balanced because of, the allocation it is likely that if the last utilization if the last task has a very high utilization let us say 0.5. Then definitely if it goes to some node it will make it change the utilization to very high values listen it.

So, here this algorithm is used when the nodes individually are scheduled using e d f, we had said that is because we are not really having any other restriction here in the form of the liu Leyland or liu lehoczky's criterion.

But, you make ask a question that, why do we maintain the tasks in the queue in increasing order of their utilization? Why not in decreasing orders of their utilization? What will be the impact of that? If we maintain the task in decreasing order of utilization and then first the high utilization tasks are assigned to the nodes. Right and later the ones that have smaller utilization their allocated would not that are more balanced, it is not

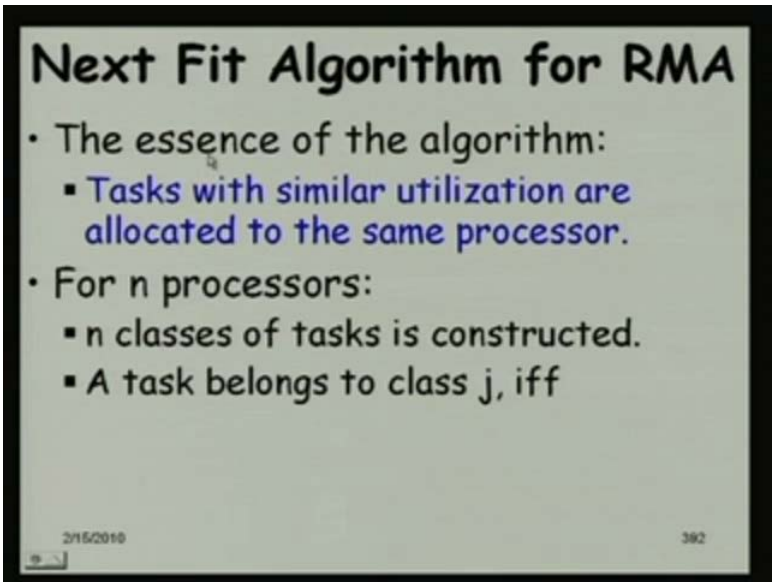
very difficult to find that out. So, I let me give it as bonus problems that please write a small program that will generate tasks of various utilization.

And then it will automatically allocate using this algorithm, one is there maintained in a queue where they are ordered in increasing order of their utilization and another one where they are ordered in decreasing order of their utilization. Please check which one because we said here, that this is the figure of merit of the utilization balancing algorithm which one will lead to more balanced utilization or will have a highest figure of merit.

What about random arrangement of task, what is the performance of that? We will have to do that for large number of task examples listen it, we have to automatically generate tasks with various utilizations not a very big task actually to check this out, and small program may be some twenty thirty line of c program which will generate the tasks. And then it will check; which algorithm will work better? Whether the variation where the tasks are maintained in increasing order of the utilization decreasing order of the utilization are random order of the tasks.

And by how much, which one will be better? On the other edge, please try to find that out and submit it to me.

(Refer Slide Time: 06:31)



Now, let us look at the next fit algorithm used in conjunction with R M A scheduling at individual nodes. The essence of this algorithm is the tasks with similar utilization are allocated to the same processor.

In my task that sees, what is the idea behind this? I mean, why tasks with similar utilization should be allocated to the same processor. I do not know exactly, what is the meaning of this heuristic? Is basically, heuristic assumption that if tasks with similar utilization are allocated to the same processor possibly, the schedulability is higher? Possibly,

But, why it is not? I do not have an answer please think about, if you find the answer just let me know. Sir, similar utilizations are equal? No, we will see similar means let us say some task has high utilization, let us say 0.8 to 0.1 let us say these are considered as high utilization. So, then they are one class of task then let us say 0.52 to 0.6 they are similar right, 0.1 and 0.2 utilization they are similar. So, for different classes of utilization we will keep a processor reserved for 0.8 to 0.9 we will have some processors 0.5 to 0.6 we will have some processor 0.1 0.2 0.2 0.3 etcetera. So, those we said that they have similar utilization.

The different classes of utilization or similar utilization we keep a separate node for them for running. But, why should similar utilization tasks run on the same processor? It is not very clear, you please think about if you find logic. If all the processors of single utilization, we have equal classes to execute? If there is a higher utilization on the lower utilization charge, that lower utilization will get extend it.

No, see as you know, that the rate monotonic algorithm are for individual nodes and here, the priority of the tasks is based on the period not on the utilization listen it. So, what you are saying quickly is not really correct. So, it is not necessary that you read the answer right now. Because, I said that it is not a very straight forward problem think of it, if you think of some good answer just let me know, it is not mandatory that you will have to answer it right now.

So, if we have n processors we create n classes of tasks for each class of task, the processors have similar utilization. We will say that, a task belongs to class j, if two to

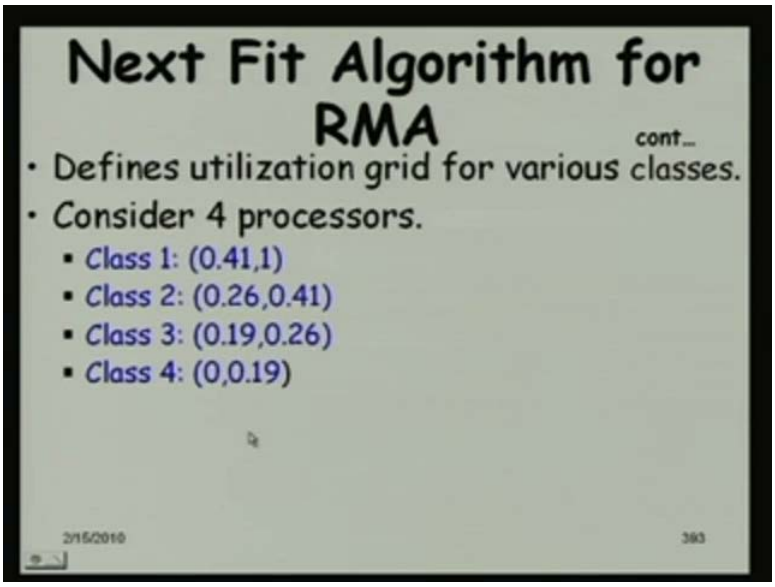
the power one by j plus one minus one is less than e i by p i which is less than equal to two to the power one by j minus one.

This term if you see, two to the power j minus one it will ring you about the liu Leyland criteria listen it two to the power one by n minus one n into two the power one by n minus one right. So, that may be a hint please try to think in this direction that for liu Leyland, task set to schedulable is n into two to the power one by n minus one possibly the developers of this algorithm those who proposed this.

They had some hunch or some hint from the liu Leyland algorithm. Where, we had the bound on the utilization for a processor to schedule n number of task is n into two to the power one by n minus one. And if we have n processors we need n classes. So, the first class will have one by two minus one right. So, that is 0.4 listen it, 1.414 listen it. Two to the power one by two will be 1.414 minus 1 is 0.4121 by 1 is 1 2 minus 1 1. So, between 1 to 0.41 that will be the first class that will be constructed listen it. So, the first one processor will be reserved for those tasks which are utilization between 0.414 and 1.

The second one second processor will be one by two plus one that is one by three. So, two to the power one by three is how much? 1.3 something I guess 1.3. So, between 0.3to 0.41 will be for the second class the third class will be one by three plus one. So, that is one by four will be something like 1.2 right. So, between 0.2 to 0.3 we will have another class that is the algorithm and the last class will be between let us say, if four processors the last class will be between 0 to 0.2 something.

(Refer Slide Time: 12:23)



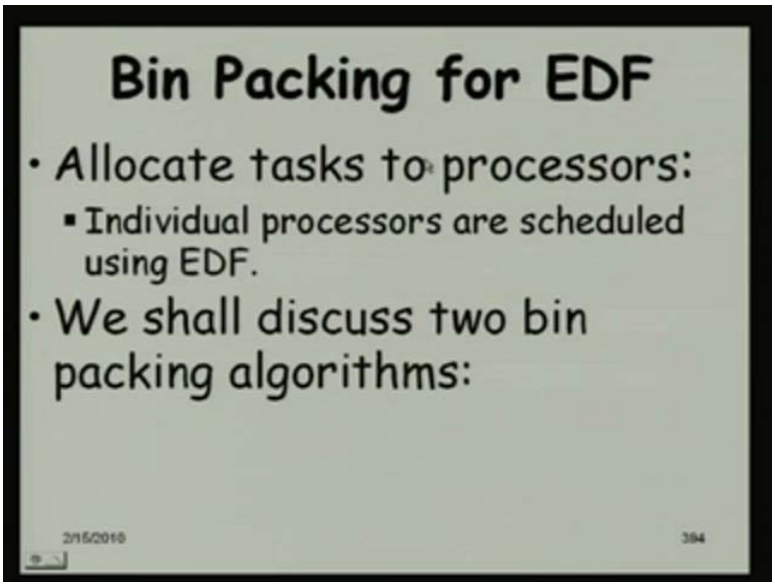
So, if we think of this algorithm it actually defines grids of utilization if it falls in some grid then it belongs to that processor. So, these are if we have four we are trying to construct four classes, the first one is between 0.41 to 1, the class two is 0.26 to 0.41,the class three is 0.19 to 0.26 and the class four is 0 to 0.19.

So, if you show this on a grid you will see that 0.41to 1 this belongs one class and 0.26 to 0.41 this is another class 0.19 to 0.26 this is a small grid here, this is another class and 0 to 0.19 this is another class. So, depending on the utilization you put them into certain class and then for each class you have one processor running those tasks.

This algorithm works well actually and requires 2.34 times the optimum number of processor. So, if you want to schedule using R M A you do not really need to balance their utilization; utilization balancing may not be a good solution for this listen it. So, if the number of tasks is small through and you can write a program which will try out all possible of the combinations of the tasks right.

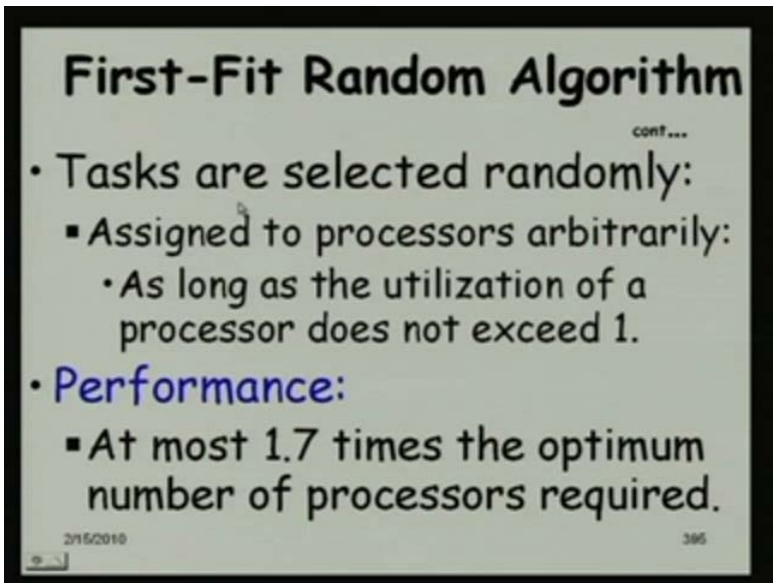
For smaller number of tasks it is possible to compute all possible combinations of the tasks and then you can see that, for the optimal one which you can compute based on all possible task combinations this solution is 2.34 times the optimum number of processors.

(Refer Slide Time: 14:35)



We have the bin packing algorithm, which is used when the nodes are individually scheduled using E D F. Bin packing is a standard problem all of you know that problem is that we have set of tasks and we just keep on assigning there and to the different processors just like you have the bin packing algorithm.

(Refer Slide Time: 15:21)



We will discuss, two variations of the bin packing algorithm one is called as the first fit random algorithm and the other is the first fit decreasing algorithm, in the first fit random algorithm the tasks are selected randomly and they are assigned to the processors also

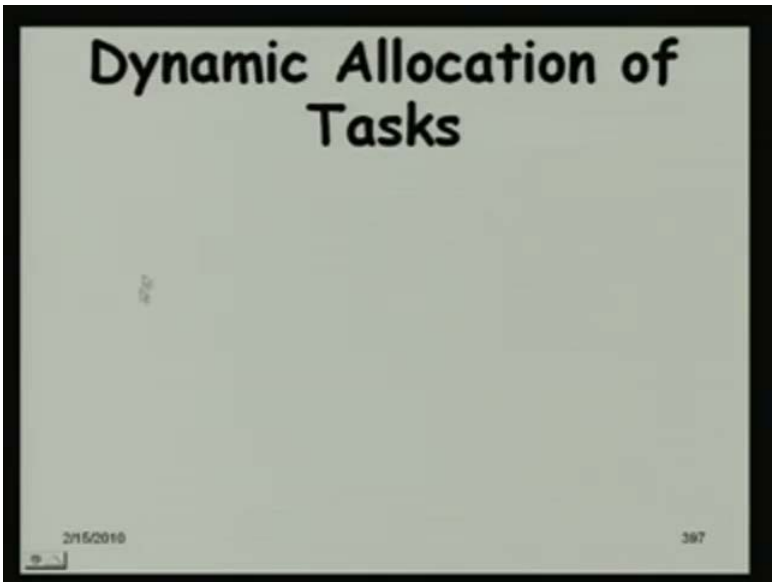
arbitrarily. So, they we will keep on assigning to the processors as long as the utilization of the processor does not exceed one.

The performance performs easy to find through a simulation experiment at most 1.7 times the optimum number of processors required. The first fit decreasing algorithm here, the tasks are started in non decreasing order selected on by one assign to the processor to which it fits in. And this performs worse as compare to the random algorithm. So, the first fit random is 1.7 and this performs much better compare to the random algorithm takes 1.2 times the optimal number of processor. Sir, can you explain this 1.7 time the optimum number of processor or 1.22.

See, the optimal number of processors the question somebody asked is that, what do we mean by the optimal number of processors? Listen it that is the question. So, see we have n number of tasks and some m number of processors and the problem is that we need to assign this n number of tasks divide this n number of task and assign them to the m number of processors. So, that individually the processor loads are balanced that is the algorithm.

Because this is being used for E D F and if the number of tasks is small, we can try out all the possible combinations of the tasks. Right, what if t 1 and t 2 come to task processor one and t 3 t 4 goes to processor two and t 5 t 6 goes to an all possible combinations of task partitions we can construct and assign to different processor. And check, which is the best combination of task? Which will make that load most balanced?

(Refer Slide Time: 18:21)

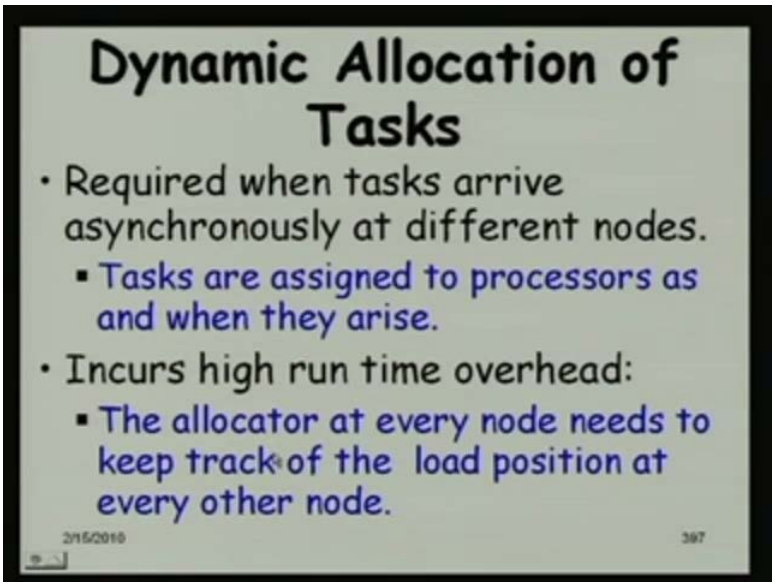


No, this not sees the individual nodes are E D F, but the allocation it is static; this is a static allocation algorithm. But then individual nodes are being run on E D F is that clear, what it is meant by an optimal solution? That we are talking of here.

Now, let us see how dynamic allocation of algorithms can be done we will look at actually, these will be required on tasks arrive asynchronously where new tasks get created a periodic task periodic task and so on. So, once the task arises needs to be assigned to a processor at that after that time till that time you do not know, what will be the state of the other processors? When this task will arise? And, which will be the best processor? We do not know, until the task actually arises and at that time we need to examine which will be the best processor on which it can run.

The dynamic algorithms of course, they handle cases which cannot be handled by static algorithms, but they incur high run time overhear, because the allocator component here has to keep track of the load position of every node based on that it will make the decision, what is the current load of different nodes needs to keep track of and based on that only it can decide where the task will run.

(Refer Slide Time: 19:42)

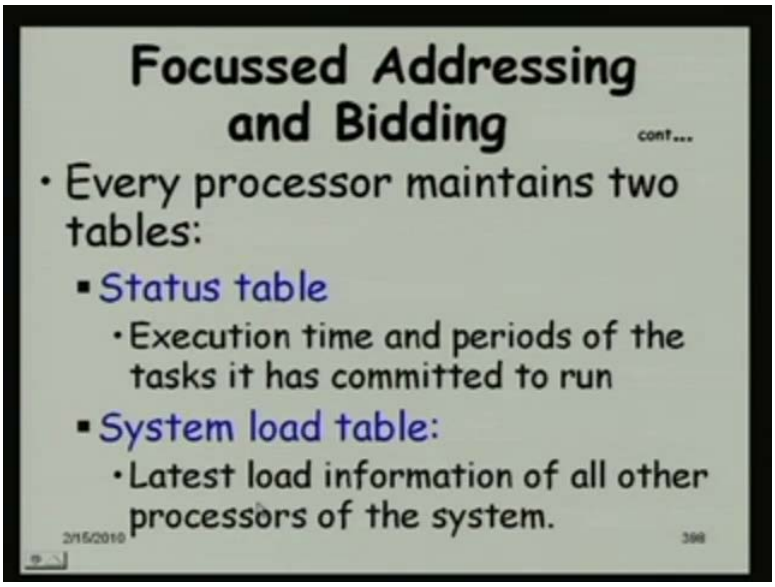


So, here the algorithm is as follows every processor will maintain two tables. One table is called as status table which is the tasks it keeps track of the different tasks it has committed to run, the execution times and periods of these tasks. So, it is about its own status the status table of a processor is about its own status.

The other table is about; the system load table is about the status of other processors in the system. So, one table it keeps track of what are the tasks it is committed to run and the other is about what is the latest load position of all other nodes in the system and of course, as we are mentioning that this would not be the current load position it will be slightly still absolute information.

Because this will be obtained through some broadcast and the broadcast cannot occur every instant listen it. It will occur once in a while and based on that it will update its table and that load position it will assume until the next broadcast comes. So, in between two broadcasts the load table will slowly become absolute.

(Refer Slide Time: 21:25)

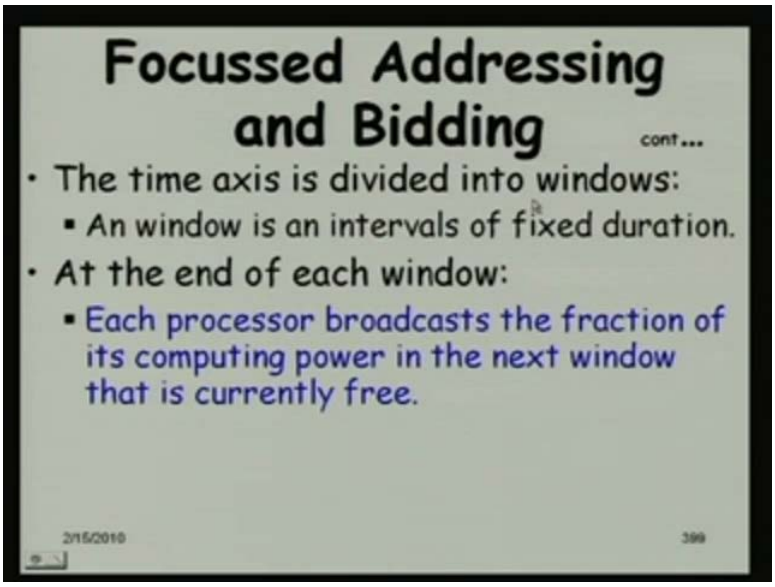


The time axis is divided into windows and the windows are basically fixed intervals of time and at the end of each window each processor will do a broadcast. They will examine their own status table and find out that for the next window, what is the fraction of processor time? That is free at that instant, so at the end of every window each processor will examine their own status table find out that in the next window, what is the fraction of computing power? The processor time that is free.

One thing of course, you are implicitly assuming here, that windows are the same for all processors. I mean see the time that every node has is the same time if the time reading is hundred in one node the all other node the reading is hundred.

But as we will shortly see that assumption is not very correct actually, the nodes will differ by some extent the times of different nodes will differ by some extent. And even if we try to synchronize the clocks see without synchronization of the clocks the times will be widely different; even if we synchronize the clocks still then the different clocks will be varying by some amount. But let us not get into that complicacy right now, let us assume that every node at the end of the window everybody knows when the window ends.

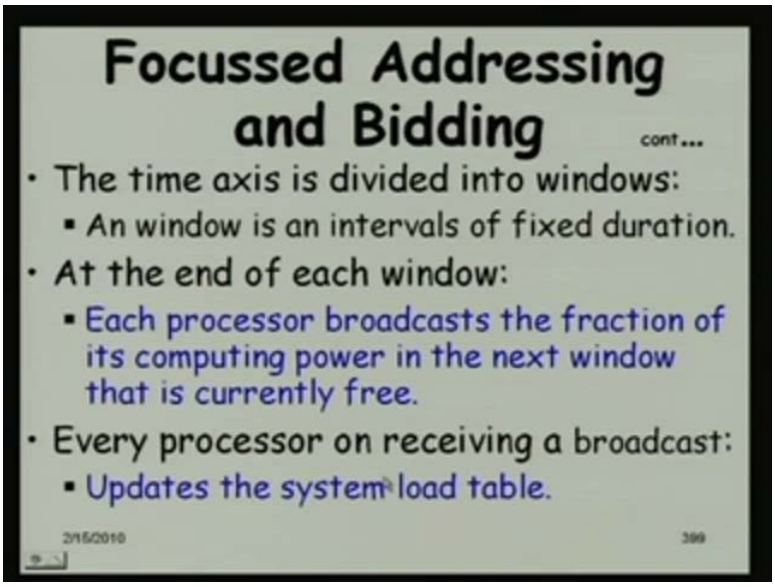
(Refer Slide Time: 23:27)



And then they examine their own status table and then they determine, what is the time that is free for the next window and then they broadcast it. And then every processor on receiving a broadcast from another node will update its load table regarding that processor.

So, just see the complexity here, that every node if there are ten nodes each node will send will broadcast to all other nodes and those nodes every node will also look at the broadcast received and it will update the system for ten what is the number of massages that will be sent you can easily compute please try to do that, how many massages will be sent in this in every window?

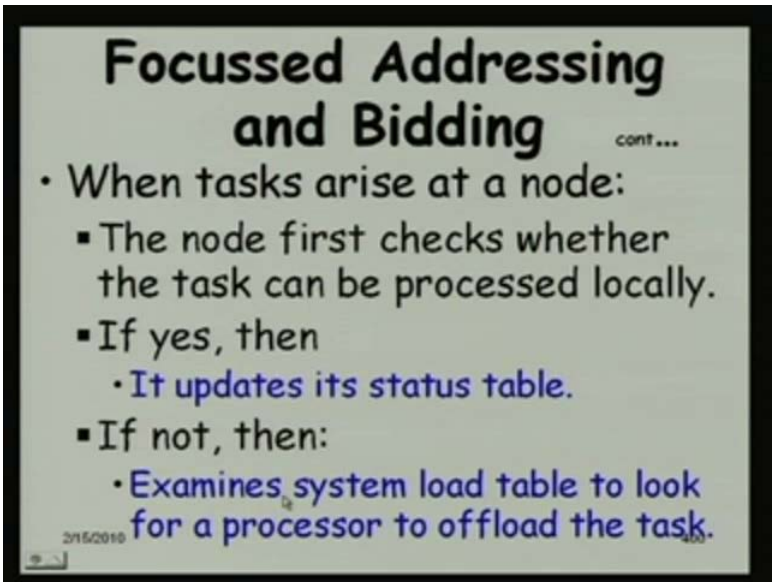
(Refer Slide Time: 24:25)



Now, the algorithm works as follows so that was about keeping track of the table right about the systems state table, now when a task arises at a node we are saying that in distributed system tasks arise independently of each other, now when a task arises at a node first checks its own status table local status table to find out; whether it can handle this so that the time constraints can be met.

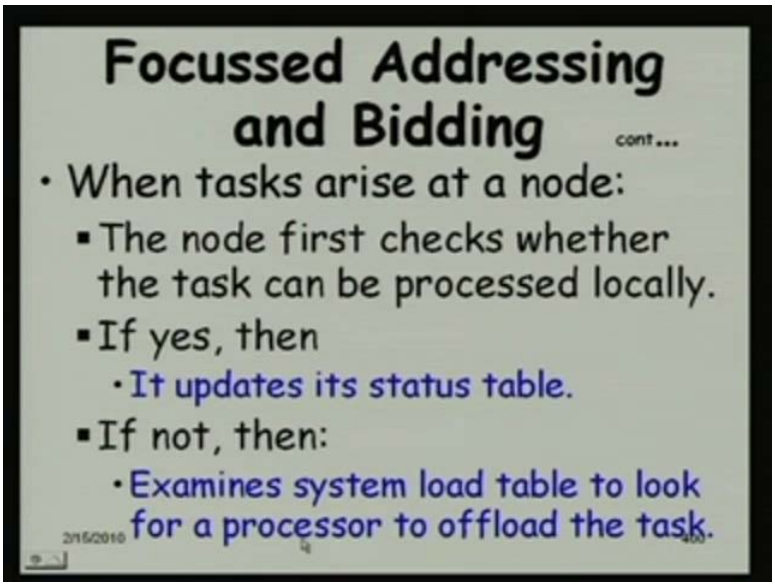
If, yes it can handle this task then it just updates its own status table, if not then it has to examine the system load table to find out a processor, which can best handle this task? and then it would have to migrate this task to that processor and of course, it has to check whether the other processor can meet the deadline of the task, where it has to assume for migration of task it takes some time.

(Refer Slide Time: 25:47)



Now, one thing is that it would not be as simple as this just find it consult it system load table and just send the task to that processor it would not be that simple, it would not work actually if we do that just examine the system load table and just upload the task to that processor, why is that? is the question clear. The question is that, if the node just examines the load table find out the processors which can handle this task that is the task can meet its deadline find those processors and to one of them just migrate the task it will not work.

(Refer Slide Time: 26:44)



Sir, there will be a time duration between that analysis and then actually migrating the task is that. No, migration you can assume a bound a time bound and that can be handled the time bound. Sir, if by the time it trying to upload other processor might have uploaded some other.

So there are mainly seeing, he says that what if two processors have that same load table and they examine that there is a lowest loaded processor and both of them they transmit decide to transmit the task to that processor. Yes, that is one problem that too many tasks might come to the lowest loaded processor the other problem is that, actually the system load table that is the global load table that has absolute information.

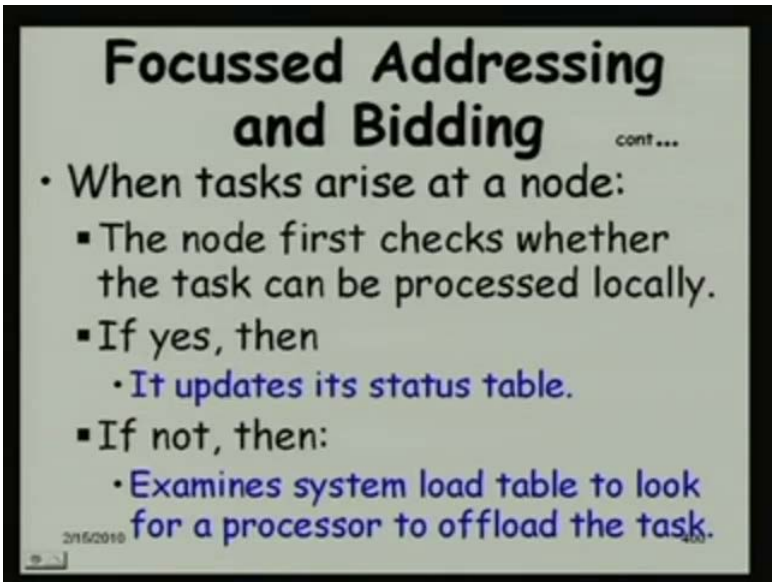
So, because if the load position changes instantly some tasks get arrive etcetera it locally there might be some task which as arrived in that node right. So, the load position that is there is absolute and whatever as per the current knowledge is the lowest loaded processor may not necessarily be true. So, that situation need to be handled we cannot directly upload the task to the lowest loaded processor right.

Sir, that table is updating the dynamic link.

The table is updated once in a while, I mean periodically after a window. The window may be let us say 10 second or something because, if you update that window if the window size is too small then too much of message complexity overhead will be there right. So, it has to be up some acceptable duration ten second or something. How do we decide this?

That depends on system to system like what the question is that, what will be the size of the window? The size of the window depends on how much time does message take place in that system, how many nodes are there? So, basically some simulation experiments have to be conducted and then found out what is a good window time. You know specific otherwise. You can say that, see this is the best window time there is no such algorithm has to be found out experimentally. So, we cannot just upload the task to a processor which can handle it.

(Refer Slide Time: 29:15)

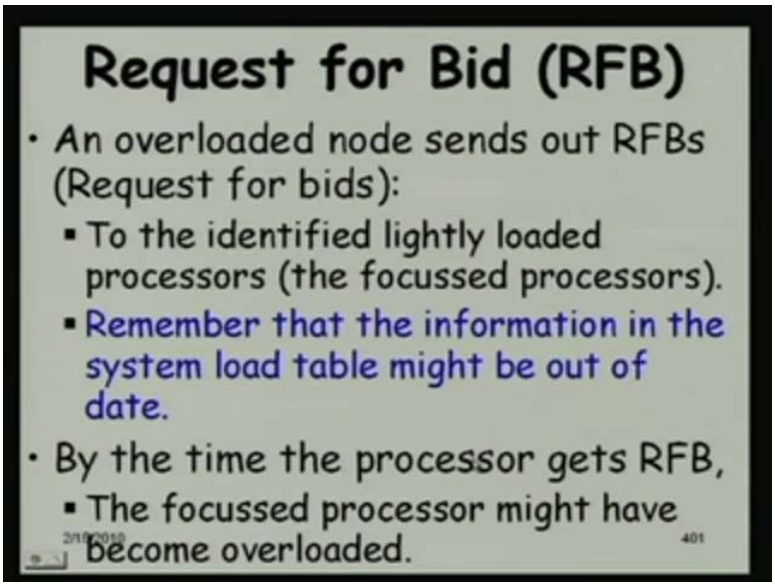


So, you need to go for a bid. So, the node which finds that it cannot handle a task sends out request for bids by consulting it is load table finds out, what are the lightly loaded processor? Which are called as a focused processor? That is the terminology used in this algorithm.

Finds out, which are the processors which can handle this load? Handle this load means the time it takes to migrate the task and plus the task execution time that must complete within the deadline of the task.

And after having identified the focused processors it sends out R F B is about the task, the R F B will contain, what is the execution time? What is the deadline? Etcetera to that task that processor and of course, by the time the R F B is arriving at the focused processors there they might have become overloaded then they will just ignore the R F B. But if the after the R F B is arrives if they find that they can still handle the requested task they will send out bids. I do not have a slide here the bidding one. So, they will send out the bids.

(Refer Slide Time: 30:36)



And once we send out bid to one processor they cannot send a bid to another processor because, they will reserve that slot for it right. So, your question regarding multiple tasks getting transferred to one node will not arrives because, once they send out a bid they will not bid for another task right.

And the processor which has overloaded which will examine the bids and then finds out, which is the best processor? where the task will most likely to meet its deadline it will transfer it there and of course, those processors which had bided at the end of the window they will again know that they have not got the bid and they will reset their load position right. Now, this algorithm incurs very high communication overhead. Why is that? Because, two components one is that the status messages have to be periodically transmitted every node broadcasts its own status and then they are updated.

The other component of the overhead communication overhead is due to the focused addressing. So, it will send out R F B bids the request for bids will be sent out to the focused processors and then the bids are met and then the task is migrated right. So, these are the complicacies in the communication two components two main components transmission of status messages and also the R F B is are transmitted and bids are sent.

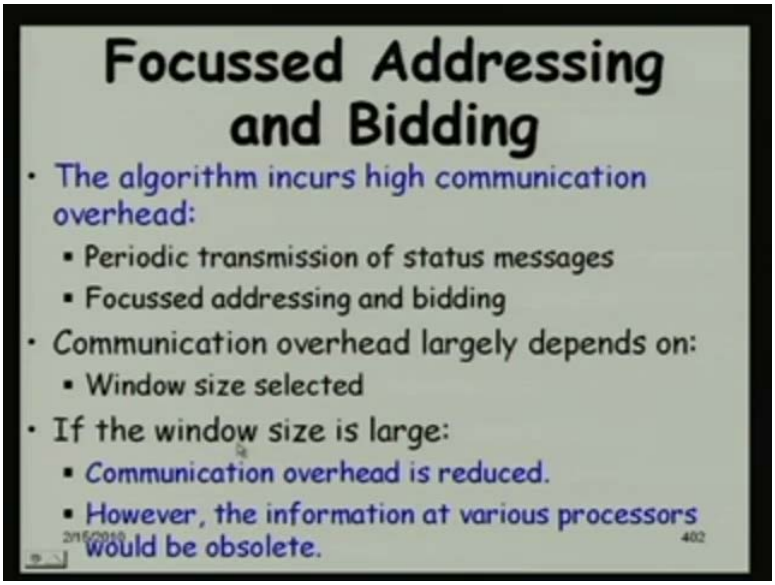
And as some of you had rightly remapped that the window size is a very critical parameter for this but we do not have an any straight forward way to determine, what is

the window size? But what we know is that if the window size is large then the communication overhead will definitely reduce, but the information will be absolute.

So, even all the processors which have been sent R F B is they might not have they might not be able to accommodate the task whereas, another processor which was having high load at that instant might have become under loaded and it is not able to transfer to that task. So, the window size cannot be too large because that load position could have considerably changed by that time the under loaded processors might have become overloaded over loaded processor might have under loaded.

And the window size would depend on several parameters; one is the task characteristic themselves like, how many tasks arrived per time? It will also depend on, what is that communication time? Message transfer time? It will also depend on how many nodes are there? Because, if there are many nodes then too much of message communication will occur due to broadcast.

(Refer Slide Time: 34:06)

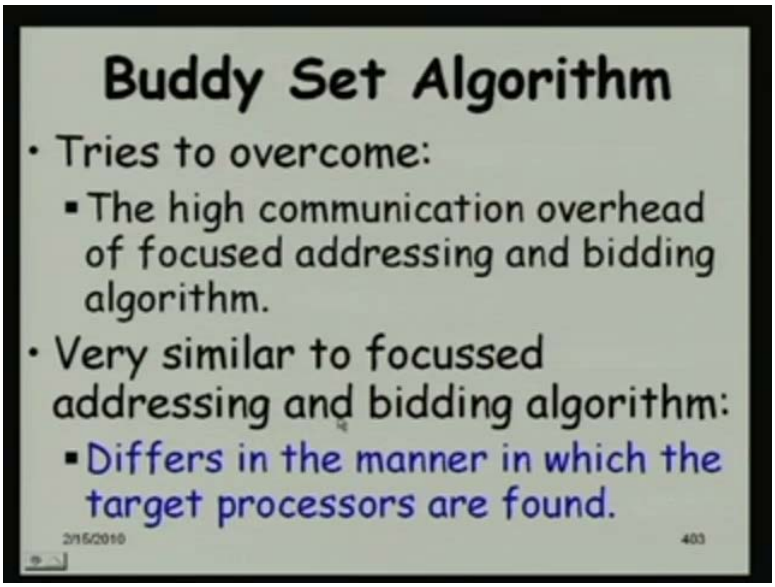


So, the focus that raised on bidding algorithm has a problem of communication overhead. So, we have this body set algorithm which tries to overcome the communication overhead problem the focused addressed and bidding algorithm its similar to that you will observe that this is very similar and working, but we will see that

the communication overhead here is much less. So, here it is very similar it also will need to keep status table etcetera.

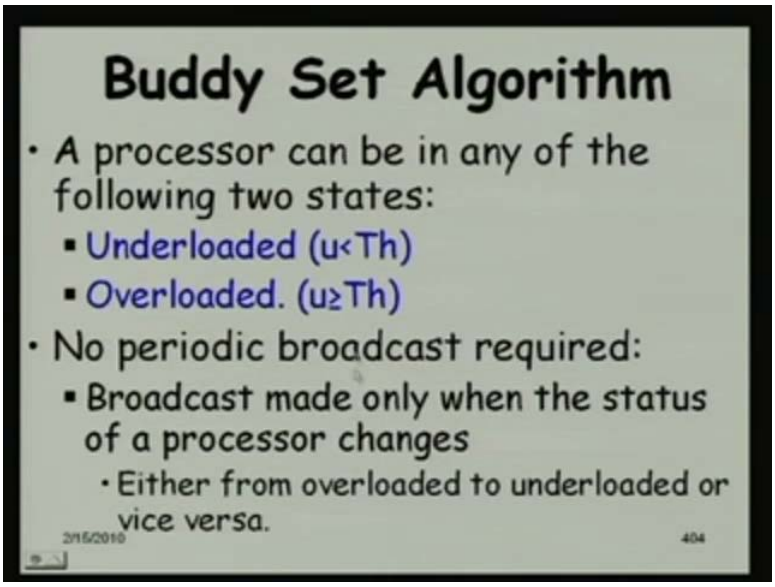
But it only differs in the manner in which the target processors are found. So, let us see this algorithm that the body set algorithm. So, here a processor can be in only two states either it is under loaded or over loaded. There are variations of this body set algorithm also in the literature, you can find that some algorithms they maintain three states; one is under loaded state one is (()) state where they are neither too overloaded nor to under loaded and then overloaded state, overloaded and under loaded these are based on some threshold values.

(Refer Slide Time: 34:52)



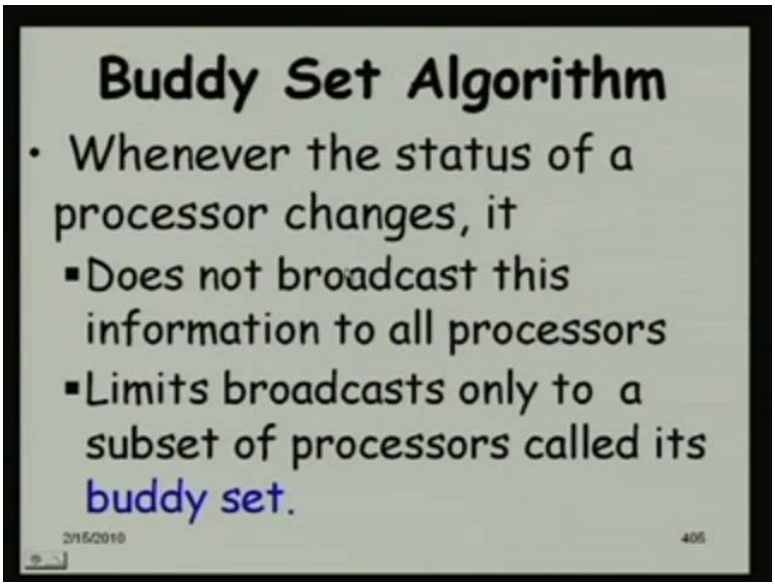
If it is more than let us say 0.4 we will say that it is over loaded less than 0.4 it is under loaded here, no periodic broadcasts are required and the node broadcasts only when its status changes form overloaded to under loaded or vice versa. So, this part of the communication is greatly reduced just check here instead of sending messages in every second or so you just broadcast only when its status changes. No need of having that window.

(Refer Slide Time: 36:03)



No, window nothing is necessary here, the messages are sent asynchronously as and when there is a change in the status. And also when the status changes it does not really broadcast to all processors it only sends it to a subset of the processor called as the buddy set. So, here also even see the number of times it has to transmit the status information is first of all reduced.

(Refer Slide Time: 36:37)

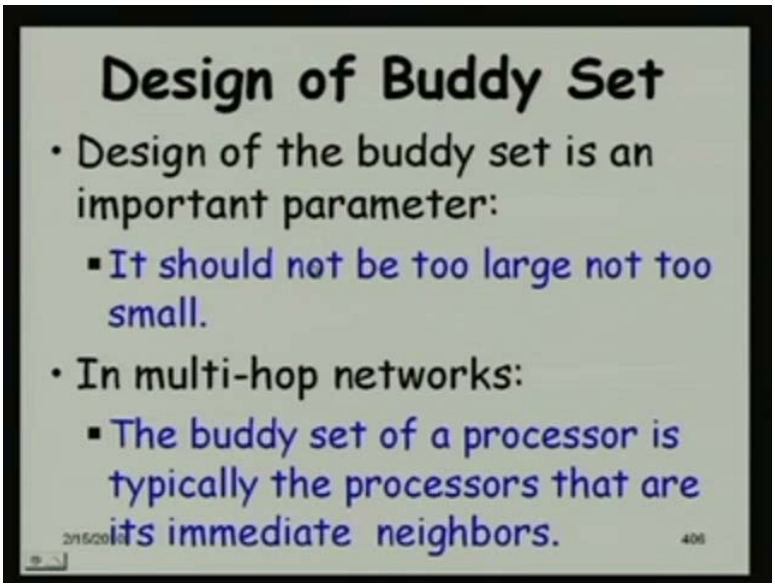


And also it is not sent to all the processors sent to only few limited processors which are called as the buddies and of course, how to find the buddy set? How to design the buddy

set? We will look at that it should not be too large if it becomes too large. It will become like broadcast if it is too small then it may not be able to send the migrate the task to a proper node.

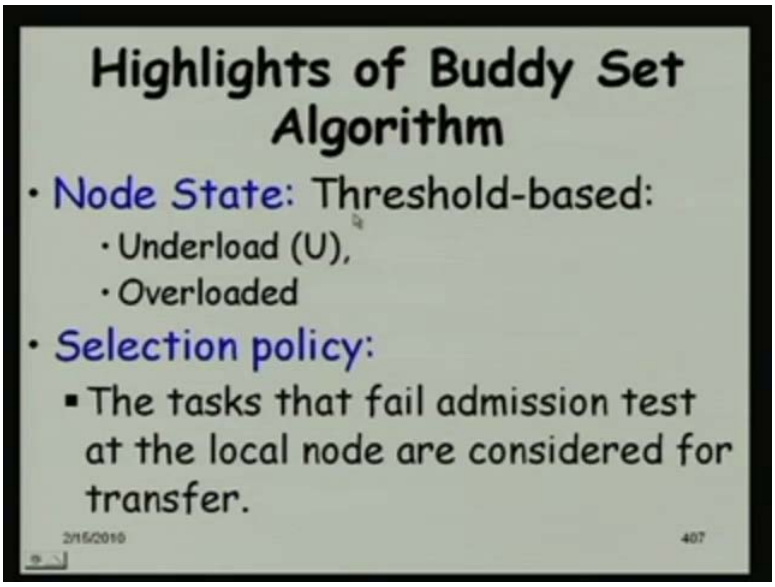
Because, it will migrate the task only to one of its buddy so there might be a under loaded processor for away, but still it does not know about that because it keeps information about only its buddies in a multi-hop network the buddy set is typically they immediate neighbors. So, this is a algorithm which is much more efficient less overhead, but the thing is that the kind of solution that is obtained by the buddy set algorithm may not be as good as that by the focused addressed and bidding algorithm .

(Refer Slide Time: 37:51)



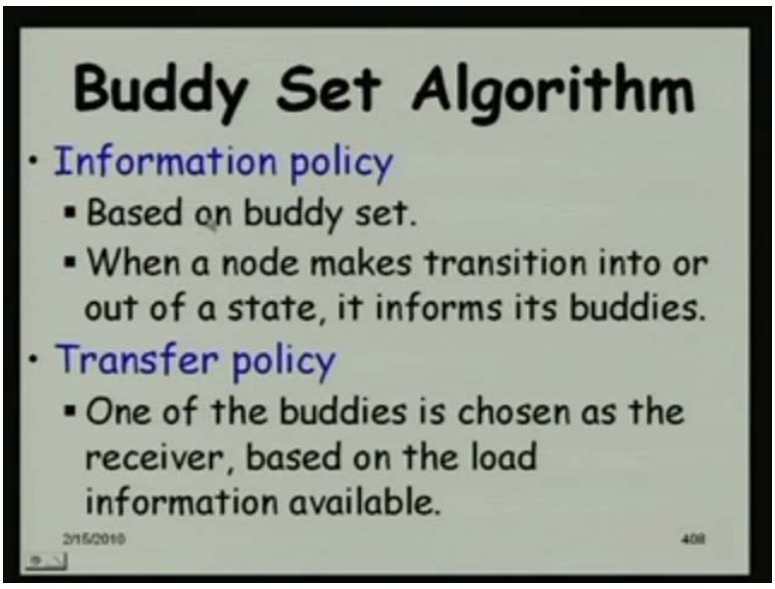
So, the highlights of this algorithms is that the node state is threshold based it does not have to keep exact utilization values only the overloaded and under loaded status is maintained. And a task on arrival is checked whether locally it can be handled otherwise it checks the information about the buddy nodes the buddy set, which are under loaded buddy set nodes and tries to transfer them.

(Refer Slide Time: 38:26)

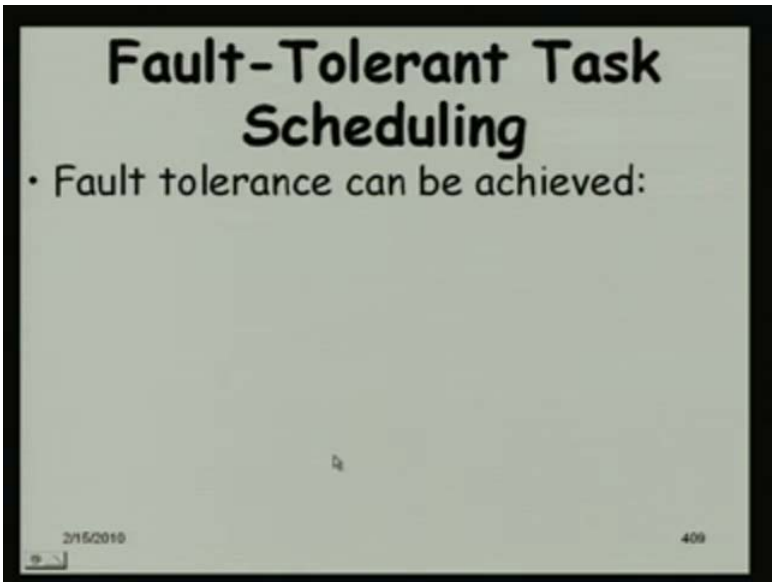


The information exchange policy is based on the buddy set and to the buddies is informed only when there is a change in the status of the node and the transfer policy is one of the buddies is chosen as the receiver and it transfers them transfers the task. Now, let us look at another category of scheduling.

(Refer Slide Time: 39:03)



(Refer Slide Time: 39:10)



Is this dynamic schedule scheduling?

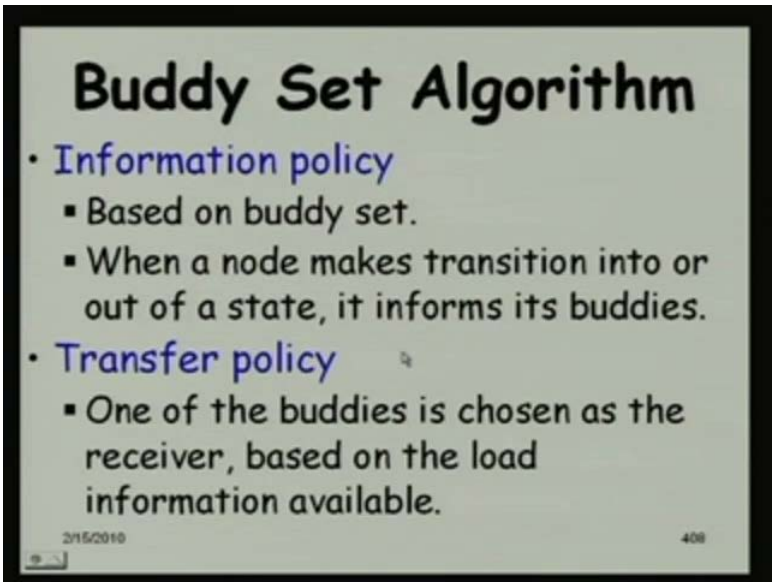
Yes.

Algorithm got their schedule?

Yes.

Whether these are applicable to some e d f or R M A or both algorithm like in static we had some (( )).

(Refer Slide Time: 39:23)

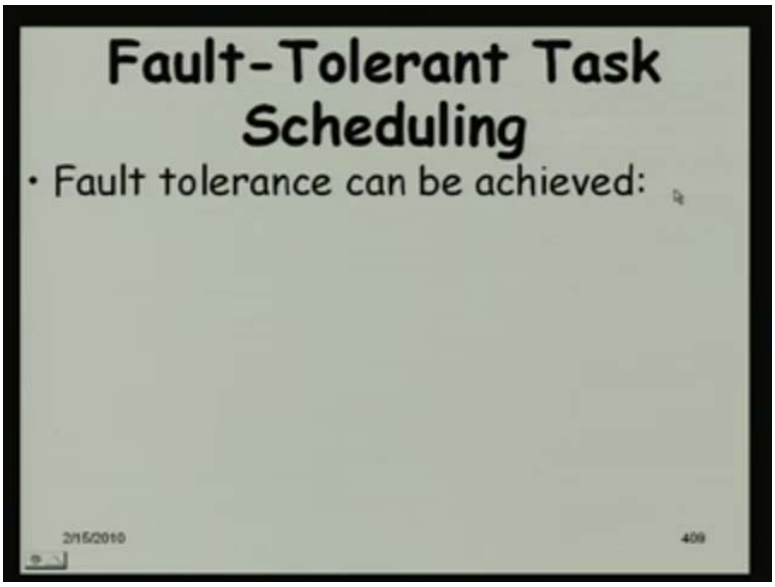


See the question that was asked is that, this dynamic allocation task allocation algorithm do they work in conjunction with some specific local node scheduling algorithm. See here, we are not constrained about a specific algorithm being used as a local node because the local node might be using R M A E D F or D M A or whatever variation we do not bother, what we are bothered here? is that whether the local node can handle it or not that is decided by the local node itself.

So, the distribution of this whether the there is a flexibility here, that local nodes might be using E D F R M A etcetera, but they have to themselves run the test and say that see whether they are able to handle the task or they need to transfer it elsewhere. So, that way this is much more powerful it that question all right fine.

Now, let us look at fault-tolerant task scheduling actually, we had said that one of the main advantage of using a distributed system is that we can incorporate fault tolerance very easily fault tolerance can be incorporated in many ways. One is have separate hardware you know separate nodes dedicated where you run the task extra copies of the task call it is a backup task you run on some dedicated node and when one node fails that takes over or there can be voting where they decide, which node has failed?

(Refer Slide Time: 41:06)



See, if cannot really decide, which node has failed? Because, it might be producing incorrect results. So, then a voting solution can be adopted, but unfortunately that solution has the extra hardware component that is built in into that solution, because we have a dedicated hardware and we are saying that the backup copies are running on that.

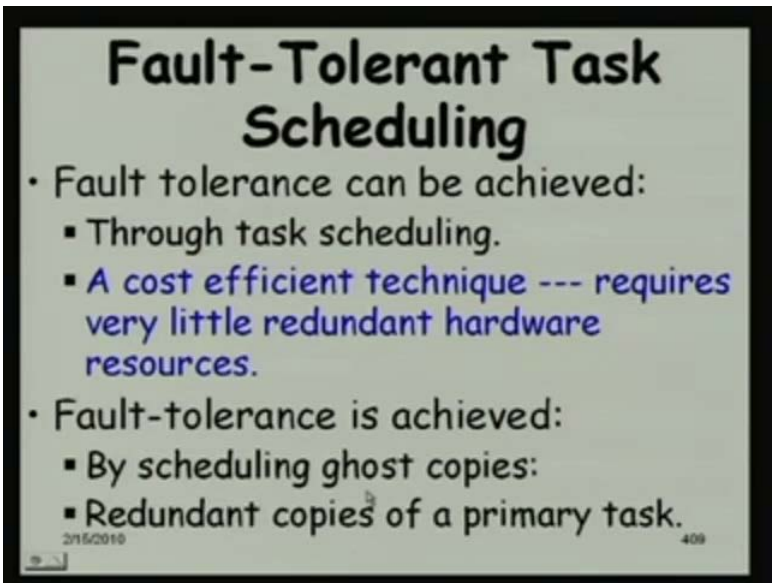
On cost consideration that is not a good solution cannot we reduce the hardware cost cannot we just run additional copies if necessary and minimize the hardware cost. So, that is about this task scheduling approach, fault-tolerant task scheduling approach, here we will not run a task unless it is required we would not have a dedicated processor just doing running this backup tasks and also we know that the failures would not be frequent just once in a while something will failure.

So, we will have multiple backups assign the same time slot assuming that at best one of them will fail and then that can run, we need not have time for all the tasks just running and wasting the resource right. That is the idea here, now let us see, how this works?

So, this is the approach of obtaining fault tolerance for a tasks scheduling rather than redundancy of hardware and running additional copies here, this is a very cost efficient technique requires very little redundant hardware resources. So, here the schedule ghost copies the ghost copies are basically backup copies these are redundant copies of a primary task.

And the redundant copies are not basically duplicate copies, these can be using different algorithm it might be more elementary solution right. Because, see once we find the primary task has failed we will have very little time to run the ghost copy actually that much time in determining the primary task as failed, we will have already passed and then these are skeletal solutions at least some result they will give.

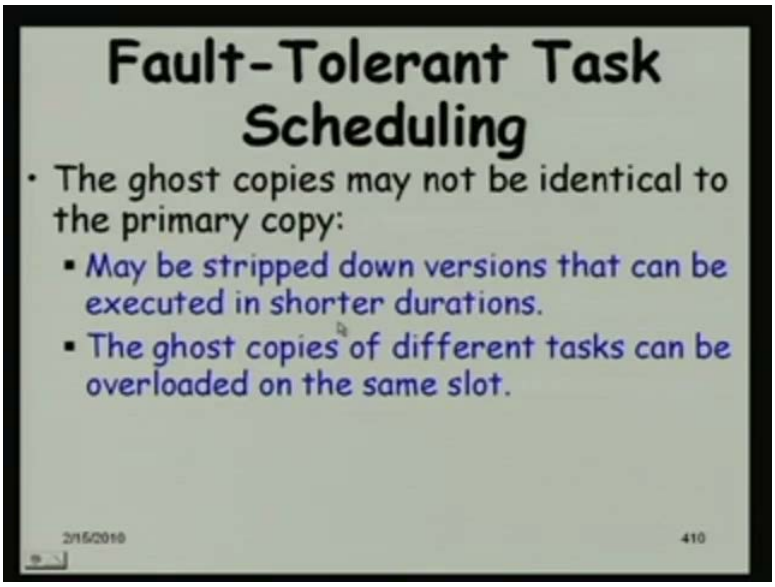
(Refer Slide Time: 43:40)



So, the redundant copies are called as the ghost copies. The ghost copies are not identical to the primary copy they are stripped down versions, that are executed in shorter duration because by the time we have determined that the primary has failed we have already adjusted some time and the deadline will already be approaching and we can only run some quick solution.

And the main idea is here, that the ghost copies of different tasks can be overloaded on that same time slot.

(Refer Slide Time: 44:16)



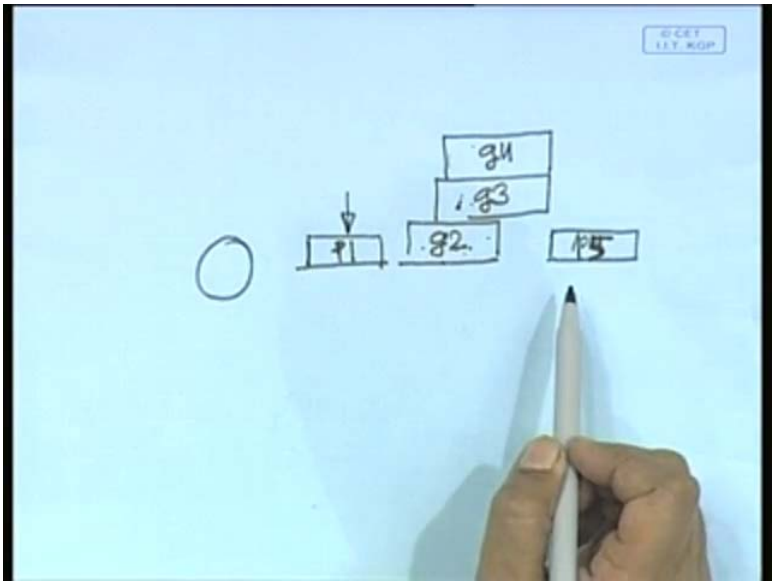
So, we can have a processor on which some tasks are running some primary tasks are running p 1 is running here, and on this slot the ghost for primary task two ghost for primary task three and the ghost for primary task four are all overloaded here, they are assigned to the almost to that same time slot here and this may be some primary four. Now, the assumption that is implicitly made here is that all the primaries p 2 p 3 p5 let me write p 5 here p 2 p 3 p 4 will not same at will not fail at the same time only one of them at best can fail.

And then, what we will do? is we will run only that ghost copy for which the primary has failed the other ghost copies will be deal located, but one thing is that the primary ghost for a primary cannot run on the same processor. Because, what if the processor fails then the ghost copy is of no use listen it. It cannot handle that fault of a processor failing it can handle of course; the task failing a software fault it can handle that this primary task had some difficulty possibly the ghost will not have that problem.

And also the other thing is that, if the primary we determine the primary will fail at it will be known that the primary will fail at some point of time then we could not have really schedule the ghost along with a primary. See, the ghost are scheduled overloaded with each other the ghosts are backed up a ghost cannot be backed up with another primary ghost and a primary cannot be allocated with the same time slot right.

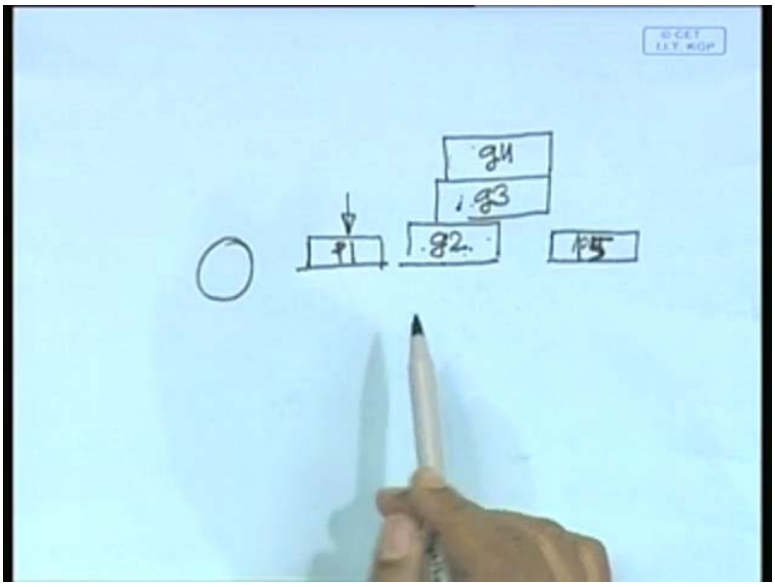
So, two constants one is that a ghost cannot be overloaded with a primary and other is that a ghost cannot run or should not run on the same processor on which the primary is running now. Should not be they get always that, ghost should not run on the same? On the same, the question is that let me get the question correctly so you are saying that, is it necessary condition that the ghost should not run on the same processor is the primary is it. So, if the ghost is made to run on the same processor is primary then some categories of faults we cannot handle.

(Refer Slide Time: 47:06)



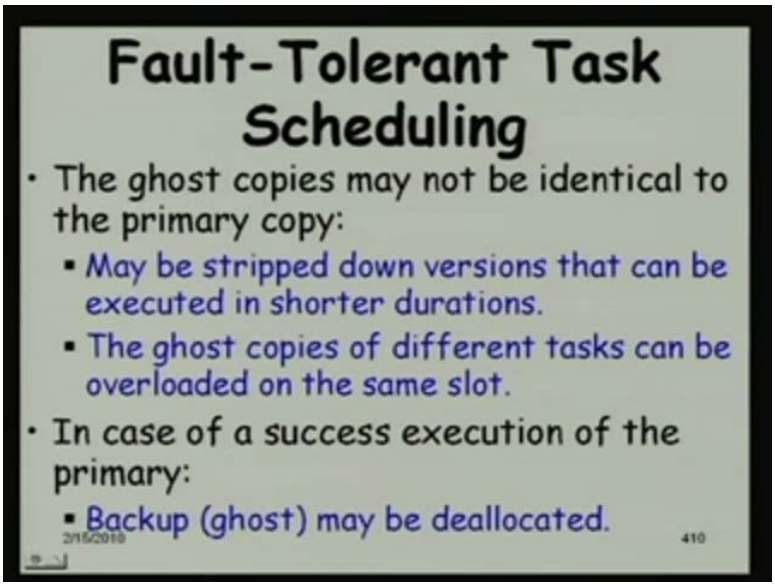
For example if there are problems with the processor. Then the ghost and primary will both fail. If it is not a hardware fault then, if it is a software fault? Yes, but only thing is that we cannot run it, before the primary completes listen it. It we cannot allocate the ghost until the primary completes. So, over loading will become a difficulty. So, unless the deadline is very long for the primary we cannot run the ghost along with the primary listen it. So, there are some difficulties here normally a ghost is allocated on a processor other than the processor on which the primary runs let us proceed.

(Refer Slide Time: 47:50)



So, if it is determined that the primary successful then the ghost is delocated the backup copy is de allocated. So, what will happen? is most of the time the ghost will get deal located. And that much time for the processor on which the ghosts are allocated will remain ideal, but see the ideal time we have reduced effectively if we had dedicated processors running the ghost.

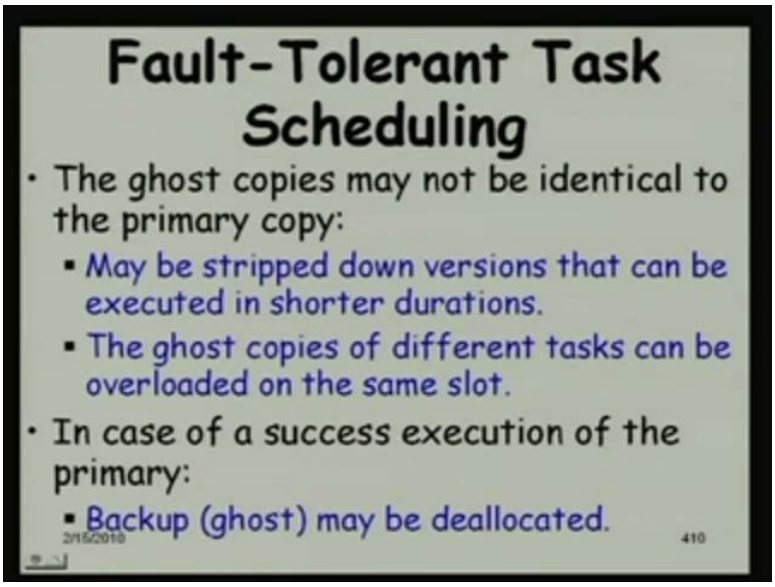
(Refer Slide Time: 48:22)



The backup copies then we would have too much wastage of those processor times, here multiple of ghosts are overloaded on the same time slot and at best we are wasting a

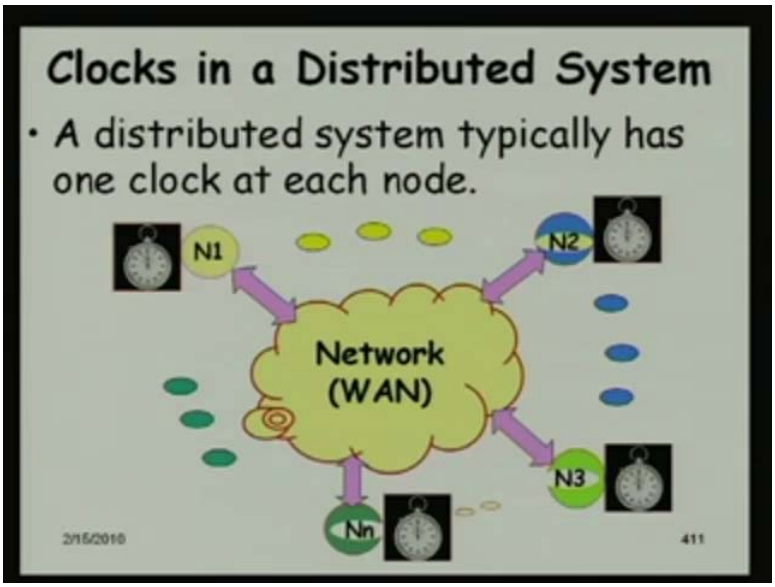
processor time equivalent to one third or may be one fourth of the time. So, this is a very promising approach to achieve fault tolerance.

(Refer Slide Time: 48:41)



Now,. So far, in all our scheduling algorithms and otherwise we assume that there are local clocks at different nodes and the clocks are all synchronized. So, each node has one clock that is true for every distributed system and we made an implicit assumption that the clocks are synchronized.

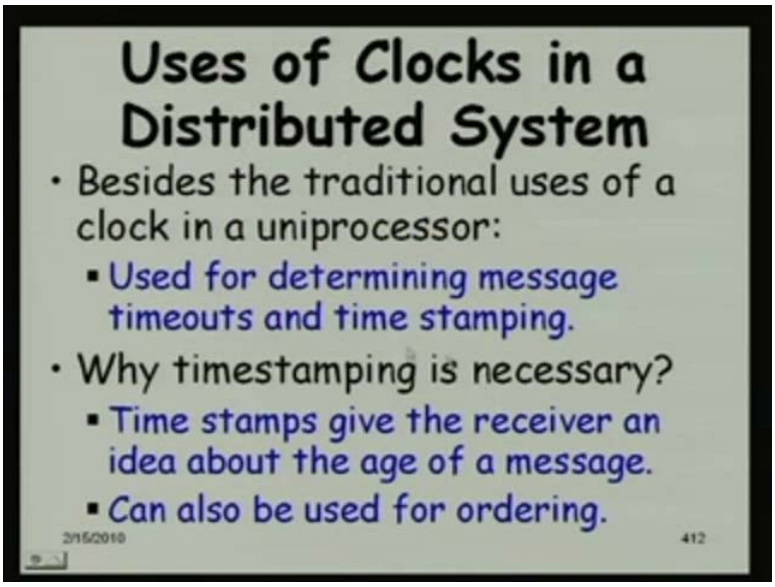
(Refer Slide Time: 49:12)



The clocks first let us see the use of this clock in a distributed system, it is used for all the purposes for which a uniprocessor uses a clock, that is for determining the task deadlines and for finding when to take off scheduling clock based scheduling all those things the clock is used, but besides that it is also used for determining message timeouts how long the clock processor needs to wait for a message and also for time stamping because, when messages are sent out? They are time stamped.

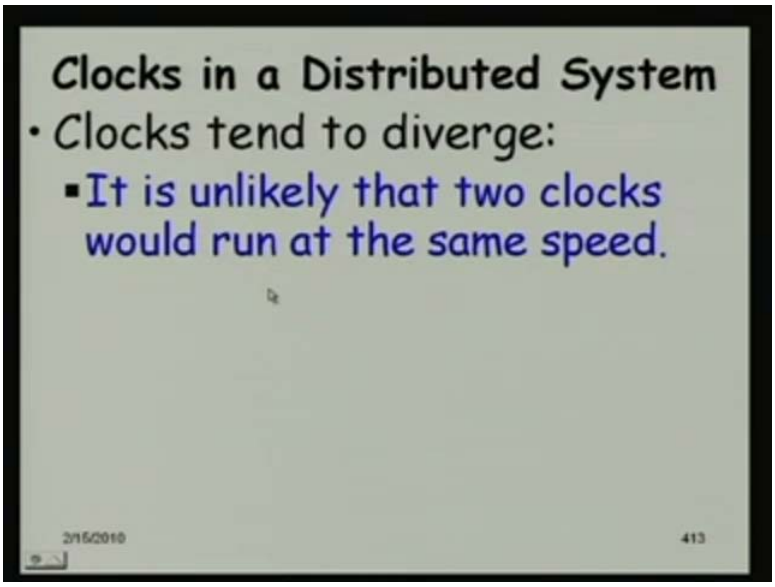
But, why do we need to do time stamp of the messages? Time stamping is done to give the receiver an idea, about the age of the message when it was transmitted. So, that can be used for ordering the messages or to determine whether the message is too absolute to be rejected. So, typically all messages sent in a distributed system are time stamped.

(Refer Slide Time: 50:17)



In any practical system, the clocks at the individual nodes tend to diverge it is unlikely the two clocks would run at the same speed, why is that? Two clocks will not run at the same speed, anybody would like to answer this question. Sir, manufacturing some difference and all there can be a great (( )). No, manufacturing means not clear actually so, first let us understand, how the clocks work?

(Refer Slide Time: 50:53)



Somebody said that manufacturing problem may be there, but first let us understand, how the clock should work? How they will derive the time? Sir, it is there are actually more crystals. Yeah these are basically crystal driven, the those principle of almost all clock even the clocks that we are using are all crystal clocks listen it. Olden days clocks are not met.

Olden days spring clock etcetera you know there is nobody uses now not even persons uses. So, about computer systems definitely not any computer system will use a mechanical spring clock right they will use a quartz clock or a crystal clock. So, then why should the time diverge? Either quartz oscillators basically, so, why should they diverge? Because of that some drift in that may it having some drift and so. No, which will have drift why should it have a drift these are all quartz clocks why should have the drift. It is can because of its properties because two things cannot be exactly a same.

Any other answer he says that two things cannot be same, you can investigate more, but the thing is that the quartz oscillation which are very high frequency oscillations they also depend on parameters like temperature of the quartz the pressure and many other parameters are there based on which the quartz frequency can vary it is heated it will its frequency might increase and so on.

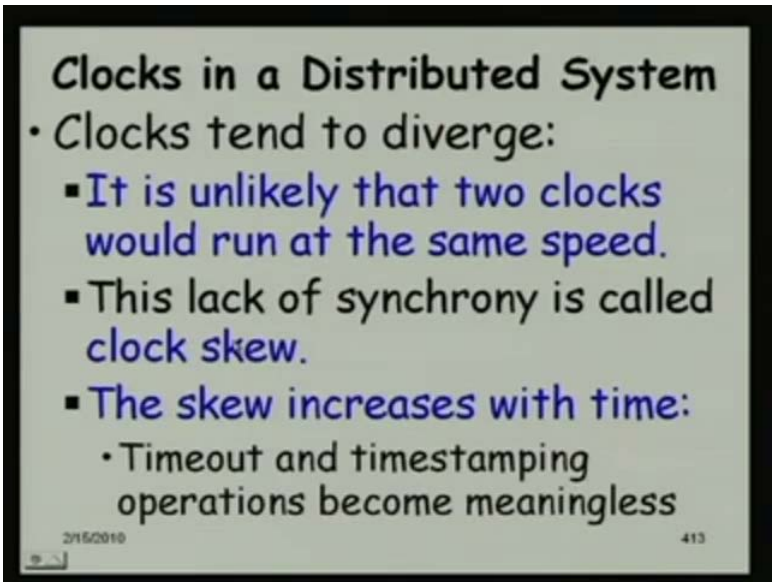
So, different places will have different conditions and the quartzes even though these are quartz clocks they will differ by a small extent and that small extent if it varies for every time pulse then it is large, because we use a large number of pulses you know giga hertz clock.

We use a large number of pulse and pulse each pulse differs by even a small amount we will have a large variation. So, the clocks they drift. So, even if we have to start with we had synchronized them as time passes it becomes one hour two hour one day the difference in the time reading keeps on increasing every time that keep on diverging. So, the clocks will diverge more and more as time passes because they run at different speeds.

And this lack of synchrony is called as a clocks skew. That is the term we will use the skew is basically, the difference in time reading between two clocks. That is, how we will define a skew? skew is the difference in time reading between two clocks and the skew will increase with time only and after they increase, see if it was a constant if the skew was constant like one clock differed from the another clock by one second we can often receive we can subtract that one second and then find out that time, but unfortunately the skew is not constant keeps on increasing.

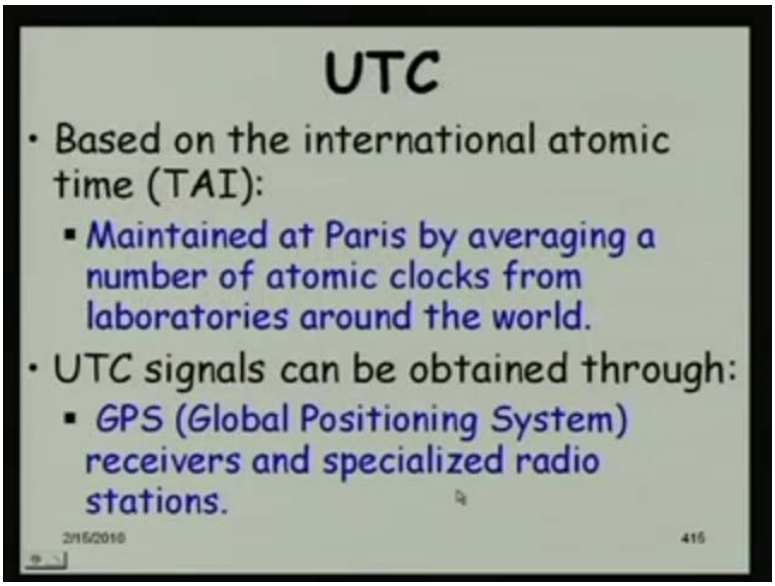
And therefore, the timeout time stamping all those operations will become meaningless because of this skew problem.

(Refer Slide Time: 54:53)



So we will discuss, how to make the clock synchronized? We will look at some algorithms, we will look at their complexity and what kind of problems.

(Refer Slide Time: 55:03)



They can handle they will not be able to handle which one to use etcetera we will look at that in the next class. Thank you.

(Refer Slide Time: 55:08)