Peer to Peer Networks

Professor. Y. N. Singh

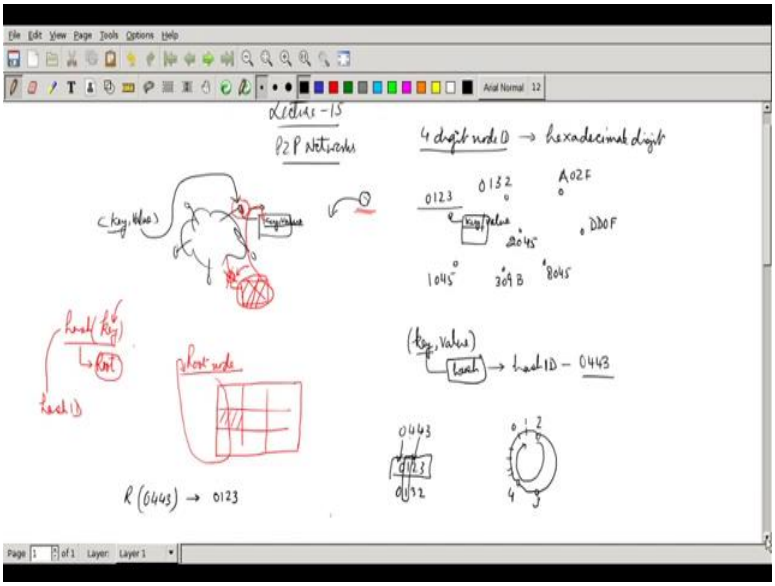
Department of Electrical Engineering

Indian Institute of Technology, Kanpur

Lecture 15

Keeping <Key, Value> Pairs at Correct Root Nodes

(Refer Slide Time: 0:13)



This is lecture number 15, for Peer to Peer Node. In this lecture, we will be looking at a problem that happens when the key and values pairs are stored in various loop nodes. When I have a network and multiple nodes in this, they are configured in a DHT format. So, the key values when it is published, this key value will go and reside with some root node.

The problem is this root node; the way we define it is that no other node is better than this; therefore, this guy has the responsibility and maintains a database where key and value pairs will be stored. But when some new node comes in, what is going to happen? So, a new node may be a better choice for this key-value pair, where this new node would have been there and when the search for the root node of the key was done, I would have ended up at this new node.

This new node is, for example, appearing here, so I might have ended up here. But now the values have been stored here, and when somebody wants to search for it, it will always come here, and there is no entry here. So, how to take care of this situation? The new node joining in is one of the problems, and of course, then suddenly this node goes away, with that all the entries saved here are also lost.

When these entries are lost, the entries should always have to be there in the DHT network. That is the condition on which the peer to peer network will work. So, I need to have some mechanism by which ensure that some other places also the entries are available. Whichever is now going to the new root node, this has become the root node because of this new node, but this guy was the root node when this was not there.

So, the entry should now move; all these entries should now move back to this guy. So, how will this be done? So, now roughly the way they happen is you always take a key, compute the hash of this, and then search for the root of this. That is the procedure which is followed. Now, this root node has to be a move to some new place. So, we can see it through an example when how this is going to happen.

This guy will figure out that I need to move or move it to is a feasible new root node. So, the hypothesis which we have is that every root node will have a routing table. And usually, what happens is how the root node identifies that it is a root node, it takes the key value, again the key here, computes the hash, and then tries to find out the root node by trying to root this; this is the hash ID.

This node will use this hash ID to look into this is a routing table, and then from this, it will figure out that this itself is the best possible choice still. That is why it is a root node; that is a test that is done everywhere. If there will be some local routing table chain, then only it is possible that somebody else can be the root node.

So, when this new node comes in, the change in routing table should happen at all the nodes whose entry, whose some of the entries at those nodes if they are supposed to come to the new node, then in all those places, the routing table update should be happening because of the addition of this new node. That is the hypothesis. And this is exactly who this actually should be done.

So, for understanding this, let us now do an example. So, the example is, so we take a node, specific nodes here. So, I take a node that is going to be 0, 1, 2, 3. I take another node; I give it a number 0, 1, 3, 2. Then I take another node, A02F. I take another one. So, I am taking; I will write all of them down, which I am taken. I have taken them arbitrarily, and then I will try to find out what happens when a new node enters.

Ultimately, we end up having these nodes and finding out what hash ID I am looking for. So, I am looking for some key, key-value pair where it can be published or a query. I am taking this key, passing through the hash function, I will get the hash ID. So, the hash ID which I am choosing now let it be 0443. So, with this hash ID, let me do the example.

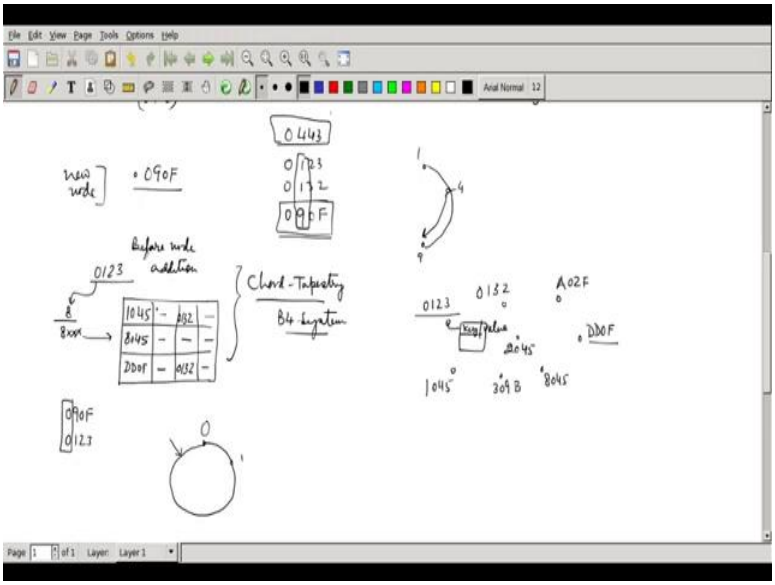
So, currently, the node list we have is the only possible nodes that will match the first ID 0. This means only one of these two has to be there, so who will be there. So, let us now compare 0443, and I have a choice of 0123 and 0132. So, the first digit is matching. So, I need to go to the second one, but the second one is only this 1 as an option. So, there will not be 4 as to go to any one of them nearly.

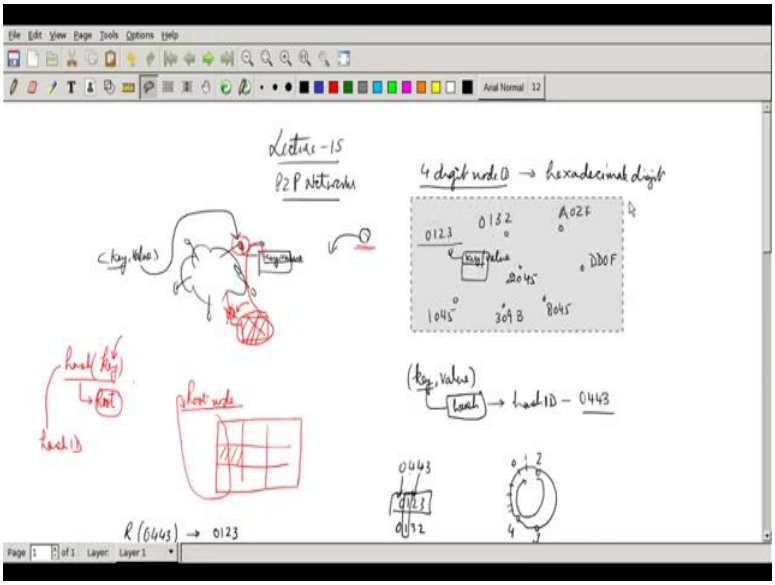
So, let us now match with the second one to decide which one it will be going. So, 3 and 2 are the only two numbers which are available. If I draw, so 2 will be here, and 3 will be here, so 4 will lie after this. So, when I am going in this fashion, there will be somewhere a 0 will be going, going to be there, but there is no node with that 1 and so on. 4, 5, 6, so this is taken, I am taking hexadecimal digit here. So, in this example, let me, I should have stated that.

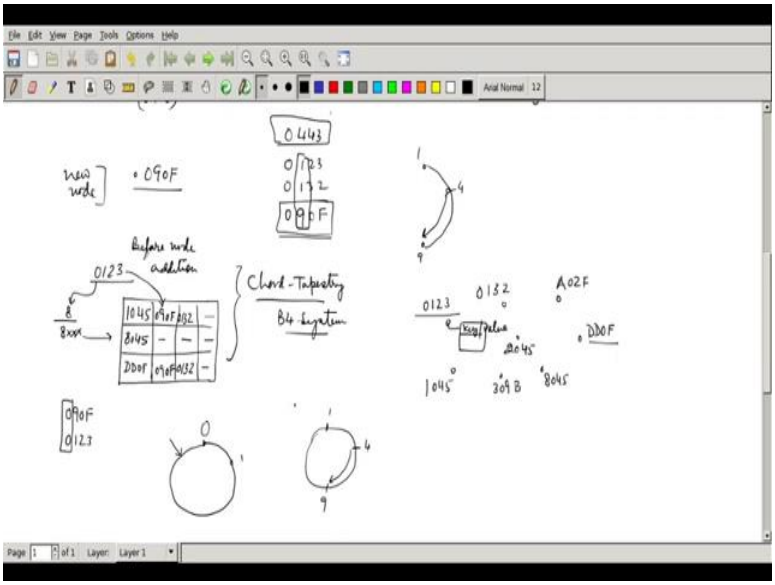
I am taking the one with the 4 digit node ID, and each digit is a hexadecimal digit. So, that is the case which I am taking. Now for 4, the next higher node is going to be 2 in cyclic fashion because I am going to go clockwise. So, 0123 obviously will now be the root node for this.

So, for 0443, if I have to find out the root node for this, this will be 0123. So, 0123 will have a local table where the database will be there, an index will be there, and the key and value will be stored. So, the key and value will be stored here. Now let us see what is going to happen if I introduce another node.

(Refer Slide Time: 08:18)







So, let the new node which is going to be introduced is going to be 090F. So that is a new node that is coming in. With this new node, we can see that 0443. If I am going to talk about the second digit, now we have an option of 0443, 0 digits which are matching are only with these two guys, 0132 and 090F. In this case, I have to figure out when the second digit, if I make a circle, will be 1, there will be 9, and the 4 will be coming somewhere here.

This is not a node; this is a hash ID. So, this 9 will be responsible; this 090F has to be the root node for this. So, is the routing table for 0123 changing, 0123 originally is the root node, is the routing table changing? So, let us look at that. So, 0123, the routing table will look something like this. There will be 3 rows and 4 columns. Obviously, in 0123, you have to look at all the nodes.

So, which node I can put here so after 0, the closest one is 1, so we have an entry with 1, I can write 1045. So, this is. You can see 1045 is available here, so I can take this whole thing down or simplicity. I should write 1 here. So, 1045 will come here, which is the next entry. Now, what is going to be its predecessor? So, what is lower than 0 is a cyclic order, so you have to take 0.

From 0 you go to 1, there is a full circle, the node that will be closest here. So, in this case, this is DD0F, which will come. So, I need to put DD0F. Again, I should have commented that the routing table I am using is for the Tapestry hybrid, which I had discussed earlier. We have been using this in our B4 system, which we are designing, working on my left.

And then, of course, which what will be the entry which will come the midway. So in 0, I can add whatever is a half value of 16, which is 8. So, I will end up in; I have to get something with 8XXX. We do have an entry here with 8045, so that will come here. Now, what will be the next entry? So this is, now let us look at this table is when 090F is not there, so this is before node addition.

So, there are only both entries that are starting with 0 or start also having 1. So, I need to put an entry which is different than 1. So, anything which is higher than 1, 02 could have been there. So it means these all will go vacant. So, the third entries were 01, is going to be there. Now, after 2 that 3 comes. So, 0132 will be available as a successor; there are no other entries, so the same thing will also become the predecessor in that row. For the third digit, the ring is there; it is for that, predecessor, successor and middle for the third digit.

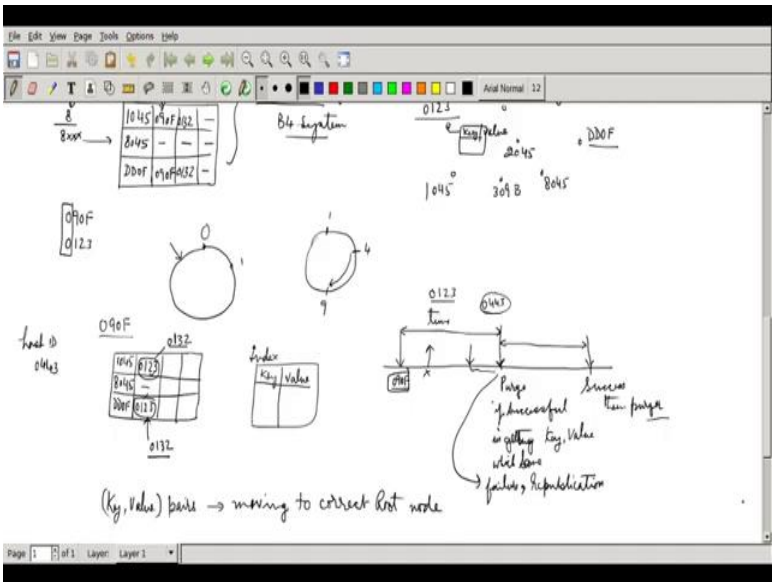
On the fourth digit, there is no match; there is no node that will be there. There is nothing with 012, so it is going to be a dash. So, this is the whole table. Now, what happens when 090F will come into the picture? It will essentially connect to somebody who will be a bootstrapping node, and with that, it will like to keep on exchanging, and ultimately, the routing table will get updated.

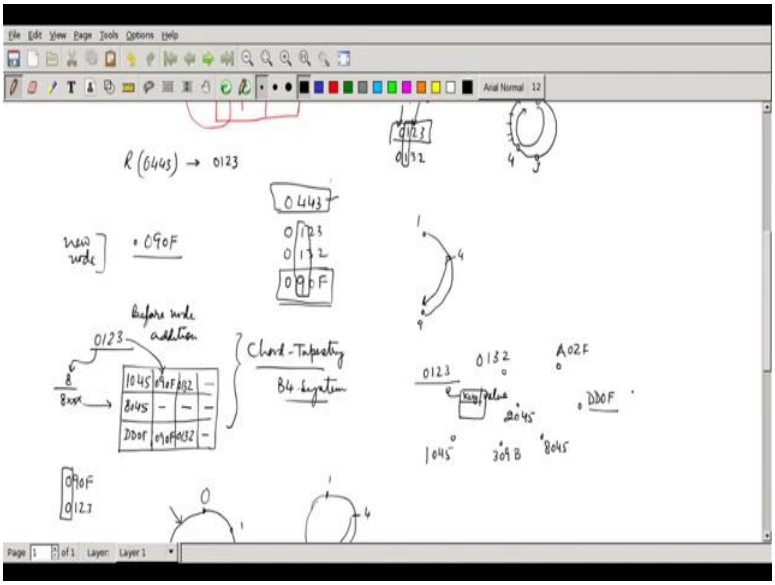
What will get updated in 0123, so if you look that 090F, it matches 0123 in the first digit, so the entry has to go in the second column. There is no entry in the second column because there is no other node with a different second digit than 0123. So there is, there is no nobody with 02, 03 and so on. So, now 090F does come. So, this will become predecessor as well as successor. There is no other option, so we can clean it off and then do it.

So, this will become 090F; this will also become 090F. So once this happens, immediately, 0123 is now figuring out that the routing table has changed, and once the routing table has changed, it should now republish its entry. So, now it should republish its entry, so where the entry will be going. So, it has to now essentially 0443; it will be searching for the best match here. So 0443, 0 first digit is matching, so it has to go to the second column.

Now you have for the second digit, this itself will be 1, and there is an entry 9, which is available. So, 4 has to be placed, so 4 is somewhere here. So, for this, the node will be now 9. So, it will just hand over, 0123 will now hand over the query to 0904. The whole entry will now be moved here, and it will be a publish request. So, when it goes to 090F, 090F will also have its routing table, which will be created over time and it will be stabilized entry. So, what will be that entry? Let us see.

(Refer Slide Time: 14:35)





Again, it will be the same 4 columns and 3 rows for 0904, 090F, sorry. The first digit, the first column, can be as it is, which will be there 1045. There is no other option. If there had been more than one option, the closer one would have been picked up in this case. 8045, then you have DD0F. The second place, it will now have only a possibility 0123 and 0132.

So, 01 is the only option, 0123 or 0132, any one of the two could have been there, so it will be keeping this. There is an alternative, you can keep 0132, or this one also will be in that case will be 0132. So, these are the two possibilities, so either this or this will be there. There is no middle entry. When 090F is going to search for the best possible node for 0443, which we are searching for, so let us look at the above; that is what we are looking for.

We are looking for this 0443. 0443 now, so hash ID is 0443. So, 0 is going to match the second column. Now you have 1 and 9 only available, it will also look at this particular arrangement, and 4 will be closer to 9. That is how we define the Tapestry. So, ultimately it ends up in 090F. So, 090F has received this entry; it will now put its index, it is going to maintain an index, but that entry will put that entry in the index.

So, now the key value is going to be updated. Now, of course, in this mechanism, there is a problem. So, 090F might come, you look at a time. So, at this time, 090F appears. Then, of course, the entry got updated, and 0123 has given it to 090F. Now should 090F should

immediately remove it. Currently, there is; I am also assuming that each node also maintains a backup mechanism. If it fails, you will look at this backup mechanism after in the next lecture.

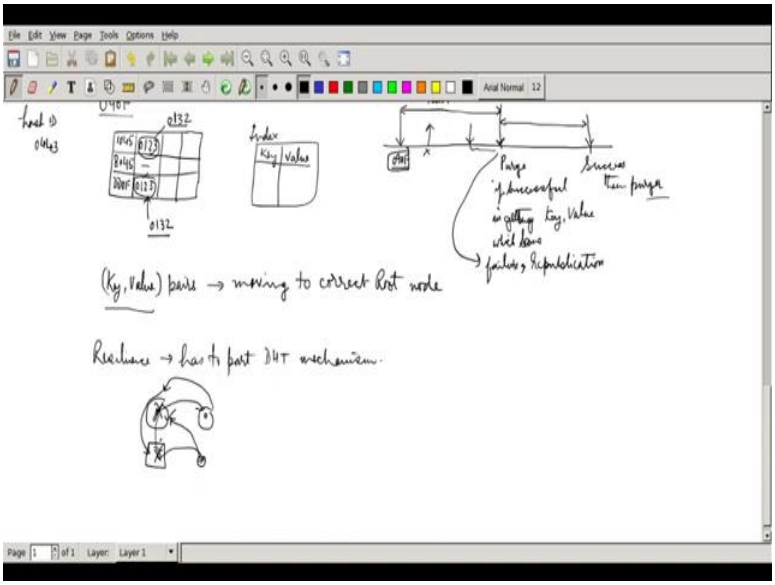
So, if it fails, the backup mechanism comes into the picture. You have given your entry to 090F, and now 090F has not created the backup so far, and it fails again, it goes off. So, backup is not there. You have already purged your entry because you are no more; the root node entry is lost. Now it comes back again; there is no way it will be able to recover the entry. Now, this is not desirable.

So, we are not going to keep this entry. We cannot simply purge. So, there has to be a timer associated with keeping the entry for that time. So, 0123 will keep the entry for this timer, and after that timer, it will check whether that entry is, if I search for this entry, if I search for 0443, whether I can get the response from 090F. If I can get it, it means this has been safely stored there, and this time is sufficient enough so that 090F has already created a backup by this time.

And at this moment, I will be able to purge. If I am successful, purge if successful in getting, successful in getting key-value pair, etc. And suppose if it can come back again by that time you find, if this scenario is going to be there, scenario 2 will be happening in this case. So, you would not be able to get it. It will be a failure when the verification happens.

So, you will again do a publication, again do a republication, and if it does not die off and next timer, the timer will be reset again. The node does not die off, this time, you will be 100 percent getting success, and then you will be doing the entry purge. This is how the thing actually should happen, and we can maintain any new node comes in, the key-value pairs will always move towards the correct root node, and key-value pairs will all be moving to the correct root nodes. The only problem that remains is now, if the node dies off, what will happen.

(Refer Slide Time: 20:11)



So, in this case, we need to figure out a mechanism for this. One possibility is somehow, the source will figure it out, and it will republish who has published this key-value pair, but that probable source is dead after putting the key-value pair. It is the responsibility of the DHT network to ensure that reliability is there. So resilience, resilience has to be part of the DHT mechanism, how that will be handled.

So, in this case, let us see how this is going to be done. We will do it in, basically, somehow we have to make sure that there is not one copy, so either root node maintains some mechanism by keeping a backup copy. If it dies off, that backup copy surface is back and then restored in whatever is a new root node; some new root node gets installed there. And this then further maintains the backup.

In case if it fails again, it is going to do this. This node comes back again; this node comes back again, then this has to go back; this entry has to be restored to this new node. Which anyway is now being done, currently, if a new node comes in an entry, has to move there that we are anyway ensuring in the current mechanism.

So in the next lecture, we will be looking at the mechanism by which this resilience will be provided. There are actually few mechanisms, few methods that can be done, and anyone can be used. In D4, we are choosing only one particular specific method to provide resilience.