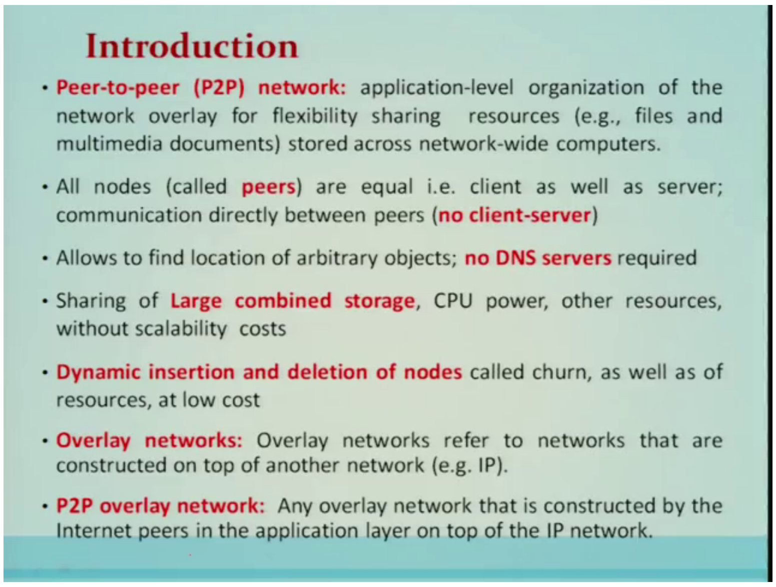
Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 19
Peer-to-Peer Computing and Structured Overlay Network

Peer-to-Peer Computing and Structure Overlay Network.

(Refer Slide Time: 00:22)



Introduction

- **Peer-to-peer (P2P) network:** application-level organization of the network overlay for flexibility sharing resources (e.g., files and multimedia documents) stored across network-wide computers.
- All nodes (called **peers**) are equal i.e. client as well as server; communication directly between peers (**no client-server**)
- Allows to find location of arbitrary objects; **no DNS servers** required
- Sharing of **Large combined storage**, CPU power, other resources, without scalability costs
- **Dynamic insertion and deletion of nodes** called churn, as well as of resources, at low cost
- **Overlay networks:** Overlay networks refer to networks that are constructed on top of another network (e.g. IP).
- **P2P overlay network:** Any overlay network that is constructed by the Internet peers in the application layer on top of the IP network.

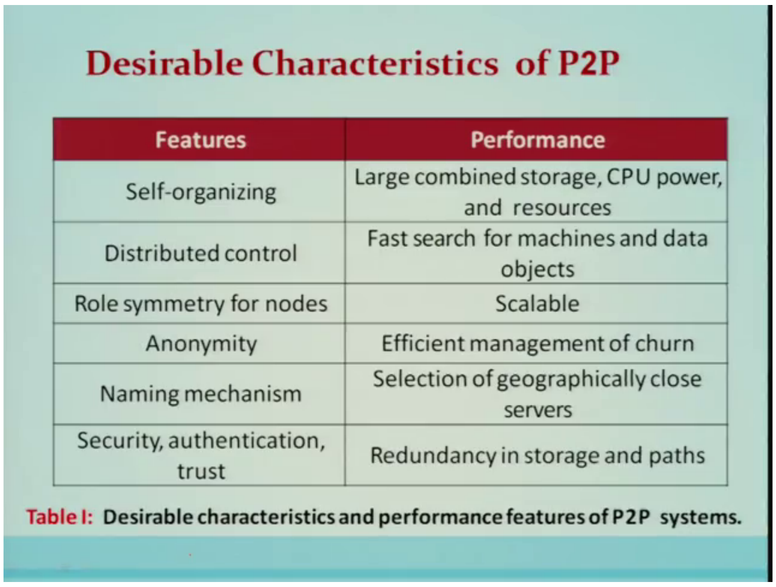
Introduction, peer-to-peer network application level organization of network overlay for flexible sharing of resources such as files multimedia documents which are stored across the network wide computers. So, peer-to-peer network will provide the sharing of resources using an application level organization which is called an overlay and this is called peer-to-peer networks.

In peer-to-peer networks all the node that is called peers are equal that is they are working as a client as well as the server and they communicate directly between the peers. Now, it allows to find the location of an arbitrary object and here no DNS is required DNS servers are required in internet to find out the location of the nodes that is nothing, but an IP address of a node and you can node and you can node name is given the DNS lookup will provide the IP address so that the network can directly access or can reach to that particular node with an IP address.

Here to find out the location of an arbitrary object here we are going to see how peer-to-peer network uses without DNS. Peer-to-peer network is basically nothing, but a sharing of large combined storage combined CPU power and combined other resources without a scalability cost. Dynamic insertion and deletion of the nodes is called the churn in peer-to-peer network as well as of the resources provided at a low cost. Overlay network overlay networks refers to the network that are constructed at the application level on the top of another network that is called an overlays of application level organization. Peer-to-peer overlay network is an overlay network that is constructed by internet peers in the application layer on top of an IP network.

So, the desirable characteristics of a peer-to-peer is that peer-to-peer networks are self organizing that is and they provide a large combined storage CPU power and resources.

(Refer Slide Time: 03:04)



Desirable Characteristics of P2P

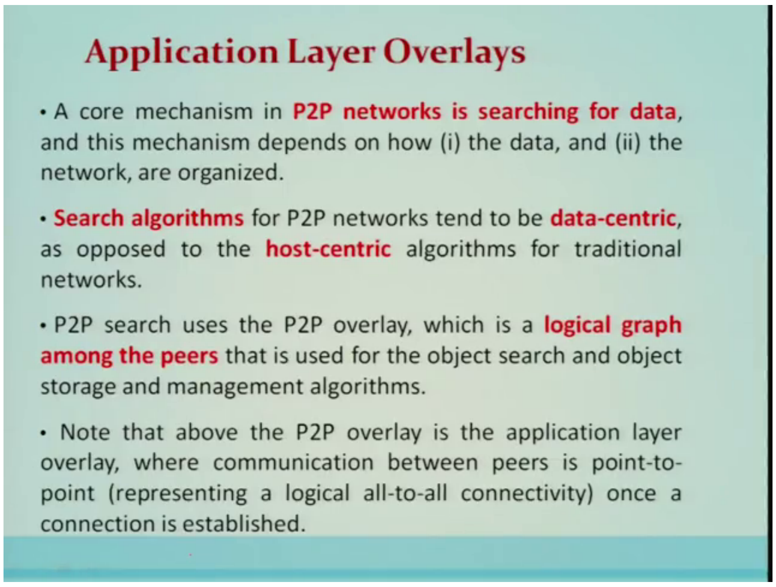
Features	Performance
Self-organizing	Large combined storage, CPU power, and resources
Distributed control	Fast search for machines and data objects
Role symmetry for nodes	Scalable
Anonymity	Efficient management of churn
Naming mechanism	Selection of geographically close servers
Security, authentication, trust	Redundancy in storage and paths

Table I: Desirable characteristics and performance features of P2P systems.

Second feature of peer-to-peer is that it has purely distributed control which will enable the first search for the machines and the object. The nodes have the symmetry as far as the roles are concerned. So, all the nodes are symmetrical; that means, they are working in the same manner and hence they will provide the scalability without any extra overhead. Another feature is the anonymity here efficient management of churn is also maintained, another characteristic is naming mechanism that is selection of geographically close servers are they being used. Security authentication and trust the redundancy in the storage and path will ensure it. So, these are the desirable characteristics and performance features of peer-to-peer networks application layer overlays.

The core mechanism in peer-to-peer network is the searching for the data and this mechanism depends on how the data and the networks are organized, so the as you know that to a search data we have to store the data in such a way that the lookup or the search for a data becomes efficient or a fast.

(Refer Slide Time: 04:18)



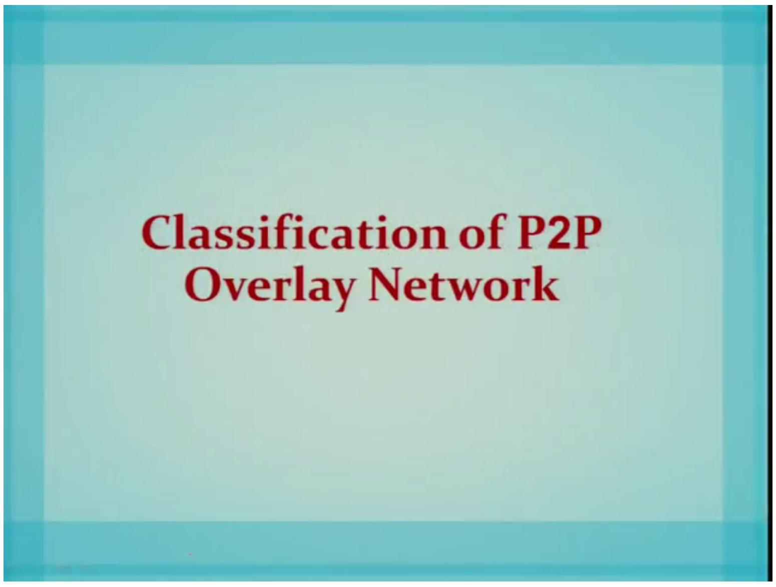
Application Layer Overlays

- A core mechanism in **P2P networks is searching for data**, and this mechanism depends on how (i) the data, and (ii) the network, are organized.
- **Search algorithms** for P2P networks tend to be **data-centric**, as opposed to the **host-centric** algorithms for traditional networks.
- P2P search uses the P2P overlay, which is a **logical graph among the peers** that is used for the object search and object storage and management algorithms.
- Note that above the P2P overlay is the application layer overlay, where communication between peers is point-to-point (representing a logical all-to-all connectivity) once a connection is established.

So, as the search algorithm for peer-to-peer network tend to be the data centric as opposed to the host centric in the internet algorithms. So, when we say data centric means the query directly can be made for a particular data or the object or a file of interest and peer-to-peer will support it.

So, peer-to-peer search uses the peer-to-peer overlay network which is nothing, but a logical graph among the peers that is used for the object search and object storage and the management algorithms. Note that above the peer-to-peer overlay is the application overlay where the communication between peer-to-peer is point to point representing the logical all to all connectivity once a connection is established.

(Refer Slide Time: 05:27)



Classification of P2P Overlay Network

The classification of peer-to-peer overlay networks.

(Refer Slide Time: 05:33)

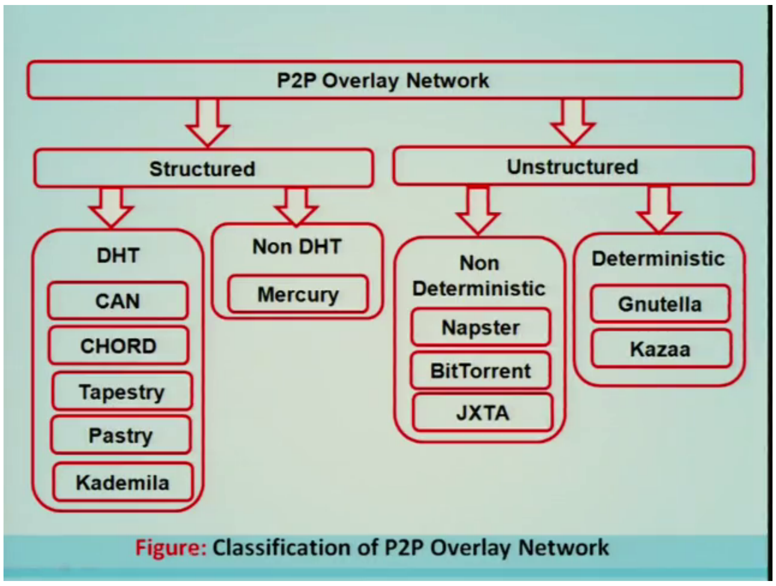


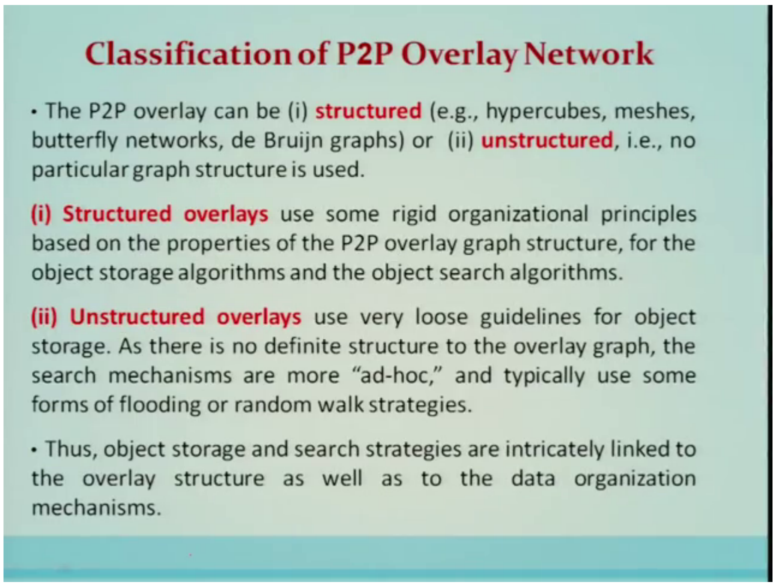
Figure: Classification of P2P Overlay Network

Peer-to-peer overlay networks can be classified in 2 types the first one is called structured the other one is called unstructured. Under structured peer-to-peer network we will see the case studies like distributed hash table can chord tapestry pastry and so on non DHT based structured overlay network is basically provided by the application which is called a mercury.

Unstructured peer-to-peer overlay network are of 2 types non deterministic and deterministic. Non deterministic are basically given as an example there in Napster, BitTorrent and JXTA deterministic are Gnutella and Kazaa.

So, the classification of peer-to-peer overlay network can be classified into structured, structured in the sense it provides a fixed topology like hypercubes, meshes, a butterfly networks, de Bruijn graphs. Unstructured overlay means that there does not have any fixed structure so no particular graph structure is used or assumed in unstructured type of overlay network.

(Refer Slide Time: 06:22)



Classification of P2P Overlay Network

- The P2P overlay can be (i) **structured** (e.g., hypercubes, meshes, butterfly networks, de Bruijn graphs) or (ii) **unstructured**, i.e., no particular graph structure is used.

(i) Structured overlays use some rigid organizational principles based on the properties of the P2P overlay graph structure, for the object storage algorithms and the object search algorithms.

(ii) Unstructured overlays use very loose guidelines for object storage. As there is no definite structure to the overlay graph, the search mechanisms are more "ad-hoc," and typically use some forms of flooding or random walk strategies.

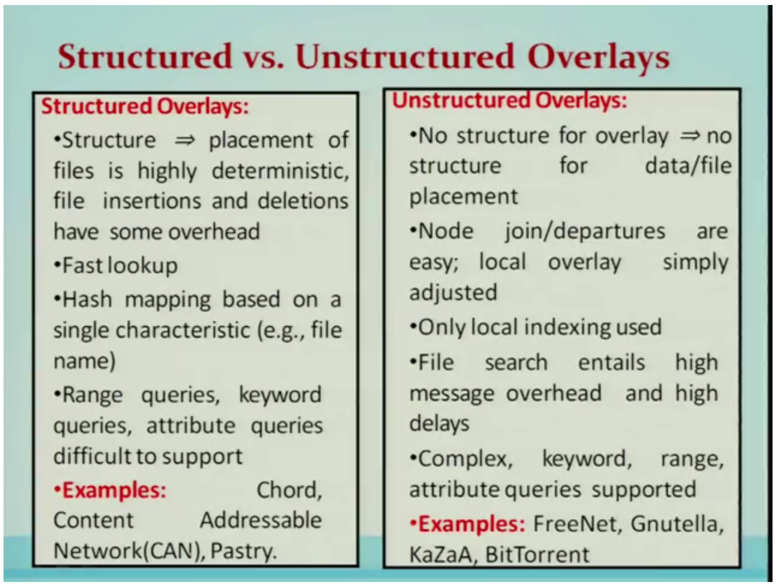
- Thus, object storage and search strategies are intricately linked to the overlay structure as well as to the data organization mechanisms.

So, a structured overlays use some rigid organizational principles based on the properties of peer-to-peer overlay graph structure for the object is storage algorithms and the object search algorithms. So, in structured overlay you can no deterministically about how the nodes are organized and where the data is stored on which kind of node in a deterministic manner.

Unstructured overlays, use very loose guidelines for object storage as there is no definite structure to overlay graph the search mechanisms are more ad-hoc and typically use some form of flooding or a random walk strategies now these unstructured overlays in turn will evolve some random structure that we will see more detail when these overlay unstructured overlays are being discussed in the further lectures. Thus object storage and search strategies are intricately linked to the overlay structure as well as to the data organization mechanism.

So, let us see the differences between structured and unstructured overlays. Structured means a placement of file is highly deterministic file insertion deletions have some overhead because of that structure. Here there is a fast lookup hash mapping based on single characteristic that is the name file name range queries, keyword queries are difficult to support in the structured overlays that is why unstructured overlays are there. Examples of a structured overlay chord content addressable network and pastry.

(Refer Slide Time: 08:24)



Structured vs. Unstructured Overlays

Structured Overlays:

- Structure ⇒ placement of files is highly deterministic, file insertions and deletions have some overhead
- Fast lookup
- Hash mapping based on a single characteristic (e.g., file name)
- Range queries, keyword queries, attribute queries difficult to support
- **Examples:** Chord, Content Addressable Network(CAN), Pastry.

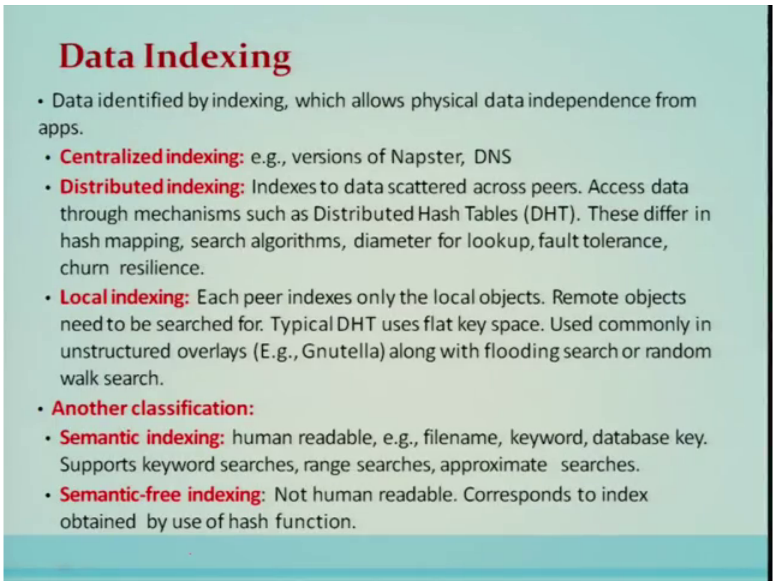
Unstructured Overlays:

- No structure for overlay ⇒ no structure for data/file placement
- Node join/departures are easy; local overlay simply adjusted
- Only local indexing used
- File search entails high message overhead and high delays
- Complex, keyword, range, attribute queries supported
- **Examples:** FreeNet, Gnutella, KaZaA, BitTorrent

Unstructured overlays does not have any structure for the overlay and the data placement or a file placement does not require any structure in this kind of peer-to-peer system. Node joints and departures are easy local overlay simply adjusted here only the local indexing is used file search entails high message overheads and high delays the complex keywords range queries are being supported here unlike in structured overlays.

Data indexing, data identified by indexing which allows physical data independence from the application. So, this is one of the most important task in peer-to-peer network how the data is organized and being accessed through the indexes which will make the physical data independence from the different applications.

(Refer Slide Time: 09:31)



Data Indexing

- Data identified by indexing, which allows physical data independence from apps.
  - **Centralized indexing:** e.g., versions of Napster, DNS
  - **Distributed indexing:** Indexes to data scattered across peers. Access data through mechanisms such as Distributed Hash Tables (DHT). These differ in hash mapping, search algorithms, diameter for lookup, fault tolerance, churn resilience.
  - **Local indexing:** Each peer indexes only the local objects. Remote objects need to be searched for. Typical DHT uses flat key space. Used commonly in unstructured overlays (E.g., Gnutella) along with flooding search or random walk search.
- **Another classification:**
  - **Semantic indexing:** human readable, e.g., filename, keyword, database key. Supports keyword searches, range searches, approximate searches.
  - **Semantic-free indexing:** Not human readable. Corresponds to index obtained by use of hash function.

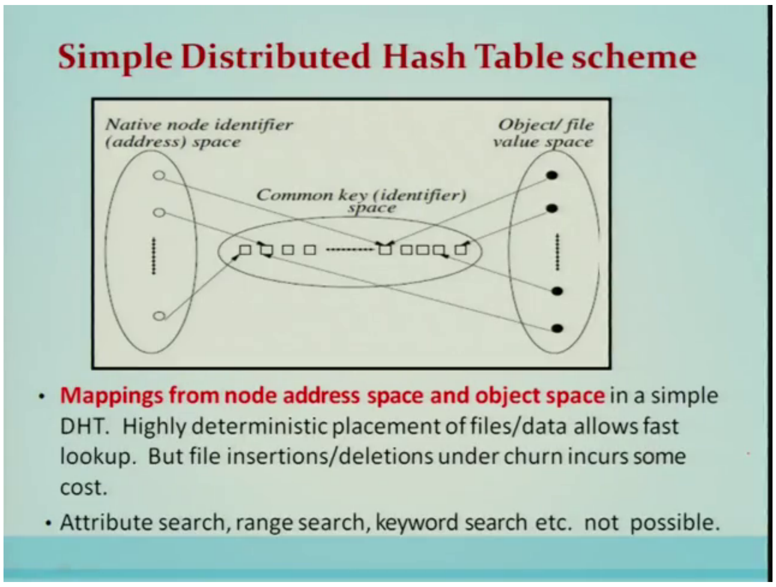
Or 3 types of indexing we are going to touch upon centralized indexing these versions we can see in Napster and DNS, distributed indexing that is the indexes to the data is scattered across the peers, access data through the mechanism such as distributed hash tables these differ in hash mapping search algorithms diameter for lookup fault tolerance and churn resilience.

Local indexing each peer indexes only the to the local objects remote objects need to be searched for typical DHT uses a flat key structure used commonly in unstructured overlays locked Gnutella along with the flooding search or a random walk search. So, these 2 are going to be discussed in more details.

So, another classification is called semantic indexing human readable filename keywords database keys are supported in semantic indexing. So, supports the keyword searches range queries and approximate searches as you know that this kind of features are supported in unstructured overlays. Semantic free indexing is not human readable corresponds to the indexes obtained by the use of hash functions, so semantic free in indexing is used in structured overlays.

So, semantic free indexing is used in structured overlays. Now, we are going to cover the structured overlays and in particular. The distributed hash tables distributed hash table scheme says that the mapping from node addresses space and object space is done in a simple DHT highly deterministic placement of file or data allows fast lookup, but the file insertion and deletion under churn incurs some cost. Attribute search range search keyword search etcetera are not possible.

(Refer Slide Time: 11:54)



Simple Distributed Hash Table scheme

Native node identifier (address) space
Object/ file value space
Common key (identifier) space

- **Mappings from node address space and object space** in a simple DHT. Highly deterministic placement of files/data allows fast lookup. But file insertions/deletions under churn incurs some cost.
- Attribute search, range search, keyword search etc. not possible.

Let us see this, let us understand this particular illustrative figure which will explain how the distributed hash table is or analyzing its common key or an identifying space. So, every node is given an id and that will form set of all ids that will form an address space for the nodes or it is also called an id-space. Now these id-space will be mapped on to a flat structure of a identifiers that is called a common key ids.

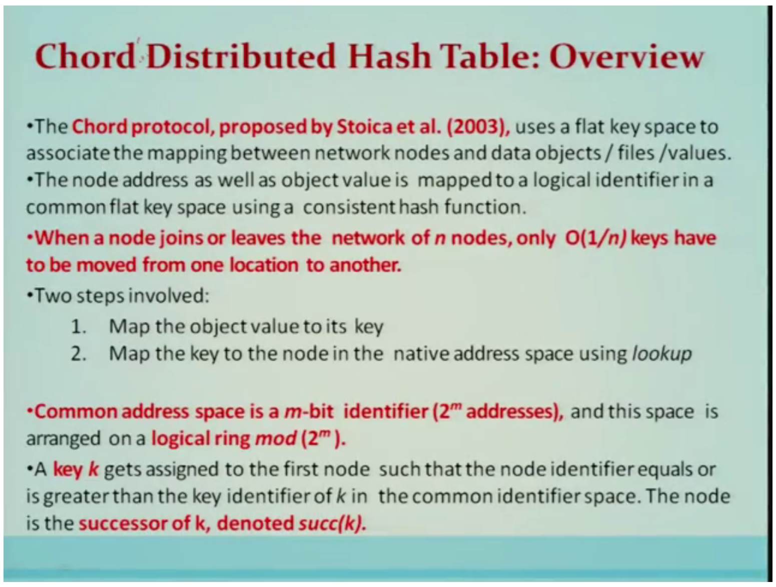
So, this mapping is done through a function which is the consistent hashing function f which will take this id of a node and maps on to a common key which is given here as a common space. Similarly the object or the file also map to the particular to the ids using again consistent hashing function. So, this particular function will take an object by name or you give a file by name and it will map to the ids which are basically in the common key space.

So, common key is space is most important and furthermore the most important part is basically the mapping or the consistent hashing which will basically support the lookup operation. So, lookup in the sense if you look up for a particular file by a name, it will basically generate a particular key and this particular key will basically nothing, but a mapping key will be mapped further to the nodes where this particular file is stored.

So, we will be looking up how the mapping from the node address space and how the mapping from an object addresses space is supported. Normally this is done through the hash table, but in the distributed environment this hash table is distributed that is why the name is called distributed hash table. So, hash table you might have seen hashing is nothing, but a supported with the hash function and this particular consistent hashing provides that to support this hash table over the distributed network and that is why it is called a distributed hash tables.

Chord distributed hash table, chord is an example which uses a distributed hash table. So, chord is a protocol which basically will support the distributed hash table.

(Refer Slide Time: 15:26)



Chord Distributed Hash Table: Overview

- The **Chord protocol, proposed by Stoica et al. (2003),** uses a flat key space to associate the mapping between network nodes and data objects / files /values.
- The node address as well as object value is mapped to a logical identifier in a common flat key space using a consistent hash function.
- **When a node joins or leaves the network of *n* nodes, only O(1/*n*) keys have to be moved from one location to another.**
- Two steps involved:
  1. Map the object value to its key
  2. Map the key to the node in the native address space using *lookup*

- **Common address space is a *m*-bit identifier (2^m addresses),** and this space is arranged on a **logical ring *mod* (2^m).**
- A **key *k*** gets assigned to the first node such that the node identifier equals or is greater than the key identifier of *k* in the common identifier space. The node is the **successor of k, denoted *succ(k)*.**

So, overview of the chord protocol, so chord protocol proposed by Stoica in 2003 uses a flat key space to associate the mapping between the network nodes and data objects or the file nodes. So, there will be a flat key space and here in one side there are the nodes which are mapped to these keys similarly there are files and objects there also to be mapped. So, that is what is written.

So, the chord protocol uses a flat key space this is the flat key space to associate the mapping between the network nodes this particular mapping so that means, a network node which is identified let us say using an IP address or a sha based an IP address. So, the mapping of this IP address into a flat keys, flat key ids and let us say it is an m-bit id.

So, hashing function will mapped and the data objects and the file also is mapped to that flat key space. So, this is the most important part of this chord protocol.

The node address as well as the object values is mapped to the logical identifier in a common flat key space using consistent hashing function that I have told you. So, the properties of a consistent hashing function is that these particular identifiers are distributed uniformly across all the nodes. So, when a node joints or leaves the network of N nodes the overhead is only one upon N keys have to be moved from one location to another location because the keys are distributed uniformly. So, using that particular property this node joining and leaving will become easy with very little overheads of moving the keys from one location to another location.

There are 2 steps involved in the chord protocol the first one is to map the object values to the keys. So, the object values that means, mapping is done using a function here the object values are given maybe a name of a file or a variable and it will give a particular key and key belongs to that flat key space. So, it is an m-bit value which will be returned. And these particular keys are uniformly distributed over the nodes.

Now, another step is to map the keys to the nodes. These keys are again mapped to the to the nodes using again a function f which will take the node id and will make a m-bit key in the native addresses space using a lookup. Now the lookup operation design of a lookup operation becomes very very important in the chord protocol.

Now, the common address space in m-bit identifier as I have been telling you; that means, m-bit identifier will be able to generate 2 raise power m different addresses and this is space is arranged on a logical ring structure that is; that means, it will support the mod 2 raise power m operations. So, that is nothing but a ring structure will form if you support mod 2 raise power m structure. This particular kind of ring structure also you will obtain if you see a wall clock in the wall clock the numbers are arranged from 0, 1 and so on after 11 it becomes 0.

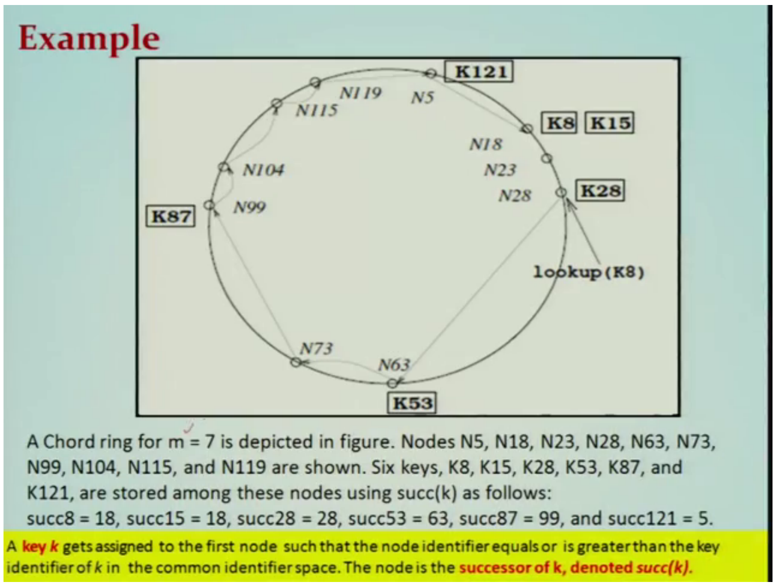
So, that mod 12 will basically give a clock structure similarly mod 2 raise power m will give the numbers which can be organized in a form of a logical ring structure that is why it is mentioned logical ring, structure will be there. Now these keys key K gets assigned to the first node here such that the node ids equals r is greater than the key identifier of K in the common address space.

The node is the successor of K denoted by successor of k. Let us take this example these are the nodes which are stored here having the keys let us say 2 7 10 and 20. So, and the keys are let us say that keys are given as 18, then here it is 5, then it is 6, then it is 9, then it is 15. Now, these keys are to be as you have seen that the key is to be located or keys to be stored in some nodes which will follow the successor of that particular key value.

So, successor of key says that either, so the key assigned to the first node such that the node ids is equal or greater than the key id of k. For example, the key 5, key 5 can be stored here in this node 7 why because the 7 is greater than 5; similarly the key 9 will be stored on 10, 15 will be stored on 20, 18 will be stored on 5.

So, using successor of K this particular storage of a key will be performed and they will follow the same common key flat structure whether it is the node or it is the key.

(Refer Slide Time: 22:26)



Example

A Chord ring for m = 7 is depicted in figure. Nodes N5, N18, N23, N28, N63, N73, N99, N104, N115, and N119 are shown. Six keys, K8, K15, K28, K53, K87, and K121, are stored among these nodes using succ(k) as follows:
succ8 = 18, succ15 = 18, succ28 = 28, succ53 = 63, succ87 = 99, and succ121 = 5.

A *key k* gets assigned to the first node such that the node identifier equals or is greater than the key identifier of *k* in the common identifier space. The node is the successor of k, denoted *succ(k)*.

The same example is illustrated over here, here this is the chord the ring of m-bit is 7. So, 7 means it will generate the keys of the range 0 to 127 values will be organized in a form of a ring structure and that is depicted by this logical ring.

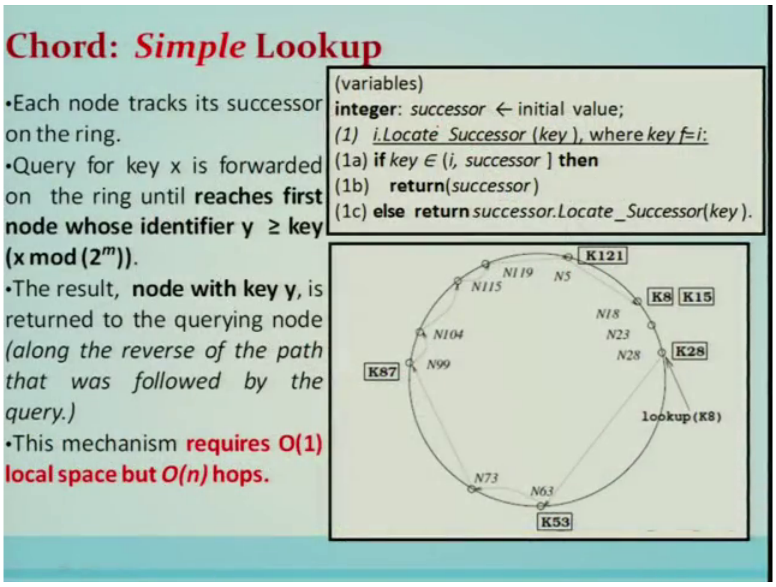
So, the nodes N 5 you can see over here. So, I will draw another diagram which will express you let us say this is the node N 5 and this is N 18, N 23, 28 and so on. They are being given an id and they are stored in a form of a ring ids will be drawn out of these set of values from 0 to 127.

Now, when 6 keys are given which are required to be stored in these numbers let us take the example of K 8. So, 6 keys K 8 K 15, 28, 53 and are stored among these nodes using successor operations as follows. So, successor of 8 here successor of key 8 will be what? So, successor of key 8 will be here in this case 18 it will be stored here in 18. Can it be stored here in 5? It cannot be stored why because it says that a key gets assigned to the first node such that the node identifier equals or is greater than the key identifier using successor function.

So, successor of K 8 is 18. So, 18 it will be stored K 15 successor of 15 is again 18, so on this particular same node 2 keys are restored. Then successor of 28 since there is a node which is having the id 28. So, it will be stored on the same node successor of 53, successor of 53 will be stored on the successor of 53 that becomes the node 63 why because it cannot be stored in 28 because 28 is less.

Now, similarly the successor of 87 is 99, so it will be stored in 99. Successor of 121 will be stored in 5 why because the mod 2 raise to power 7 operation will perform and here 121 will be stored in node number 5.

(Refer Slide Time: 25:24)



Chord: *Simple* Lookup

- Each node tracks its successor on the ring.
- Query for key x is forwarded on the ring until **reaches first node whose identifier y ≥ key (x mod (2^m))**.
- The result, **node with key y**, is returned to the querying node *(along the reverse of the path that was followed by the query.)*
- This mechanism **requires O(1) local space but *O(n)* hops.**

```
(variables)
integer: successor ← initial value;
(1)  i.Locate_Successor (key ), where key ≠ i:
(1a) if key ∈ (i, successor ] then
(1b)     return(successor )
(1c) else return successor.Locate_Successor(key ).
```

Now, having organized or stored the keys on the nodes now there is a possibility for a lookup of those values which are a stored on the nodes. So, when a application wants for a lookup of the key or a object by its name then using hash or using lookup function you can locate the node which stores that particular key and will be done in that particular manner. So, let us understand this particular aspect which is called simple lookup in chord. Each node tracks its successor on the ring, so query for the key x is forwarded on the ring until it reaches the first node whose identifier is greater than or equal to the key x mod 2 raise power m that I have explained you in the previous example.

So, this will form a simple algorithm here let us say that you want to locate a successor for a particular key, so here you have to see that the key whether the key lies between the id of a node and between the successor if it is between the successor then successor will be returned why because it will be a successor, successor will be known to that particular node i. Successor will be returned and successor knows where this particular key is stored on which node. Else if it is not lying between i and successor then what it does, it will it will give a successor to locate further successor for that particular key.

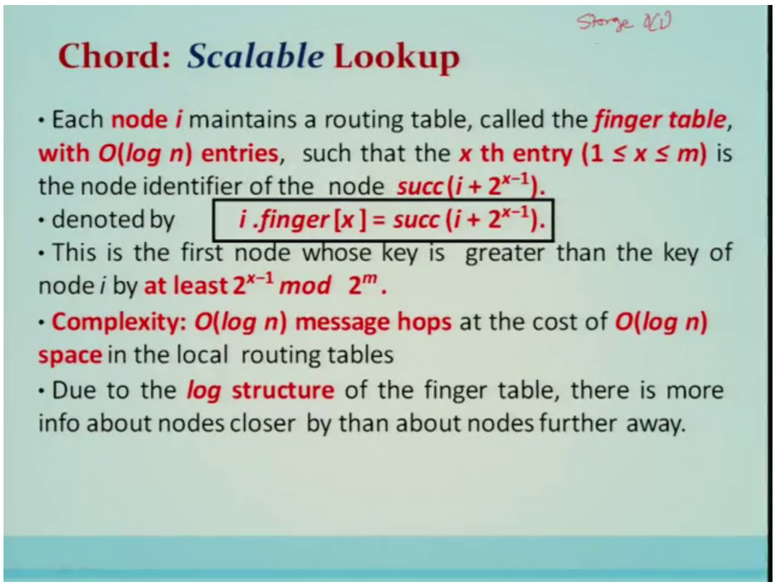
So, this particular operation in turn will be called again and again and will perform the lookup, again and again in the sense recursively it will find out. So, the example I will give here you can see there is path or a routing path it will make a routing path to reach to a particular node where that key x is stored. So, the result the node with the key K is returned to the querying node along the reverse of the path that was followed by the query this mechanism requires order one local space why because every node is storing only the successor node value that is all and, but it requires order N hopes why because it has to it has to traverse through the through the successor and will basically reach to the node which contains that value.

So, let us take the example if you let us say this example is for lookup the key value 8 at this node 28. So, lookup for the key value 8 at 28. So, i is basically 28 and its next successor is 63 and the key value K 8 does not lie between i and successor or it is beyond this particular successor K 8 value lies. So, it will reach to that particular successor and then again performs the same task whether it is between the i is becomes 63 and the next successor is 73. So, K 8 again lies beyond that successor, so it will again reach to that successor and so on, it will go on in this particular manner till it reaches N 5.

Now, when it reaches N 5, so i is N 5 and the next successor is 18 and the key which is being solved is 8 so that means, this 8 is basically greater than 5 and it is less than 18. So, this particular node will restore that particular key, so here the routing path should end and the key 8 will be given as the result of the lookup.

Now, there is a possibility of improvement why because the storage was only of the order 1 because only successor information was a stored, but as far as number of hops are concerned which was required of the order N this particular hop we can reduce by increasing the storage size and hence we are going to see a more improved version that is called a scalable lookup.

(Refer Slide Time: 30:35)



Storage $O(1)$

Chord: *Scalable* Lookup

- Each **node *i*** maintains a routing table, called the ***finger table*, with *O*(*log n*) entries**, such that the ***x* th entry ($1 \le x \le m$)** is the node identifier of the node $succ(i + 2^{x-1})$.
- denoted by $\boxed{i.finger[x] = succ(i + 2^{x-1}).}$
- This is the first node whose key is greater than the key of node *i* by **at least 2^{x-1} *mod* 2^m**.
- **Complexity: *O*(*log n*) message hops** at the cost of ***O*(*log n*) space** in the local routing tables
- Due to the ***log* structure** of the finger table, there is more info about nodes closer by than about nodes further away.

So, a scalable lookup will try to increase the storage from order 1 to order m, m is of the order log n. Similarly, the routing path or the number of hops which are need to be solve this of the order log n. So, you see that search will be reduced from order n to log n, but the storage is being increased from order 1 to order m and that is called the scale of scalable lookup we are going to see how it works.

So, node i will maintain a routing table which is called a finger table of the order log n entries, here we call it is m different entries such that xth entry of that finger table which is having m different entities is the node id of the node that is called successor of i plus 2 raise power x minus 1 and denoted by i dot finger of x is successor i plus 2 raise power x minus 1.

So, this is the first node whose key is greater than the key of the node i by at least 2 raise power x minus 1 mod m. So, the complexity t here is of the order log n message hops are required at a cost of order log n space in the routing table that I have already explained.

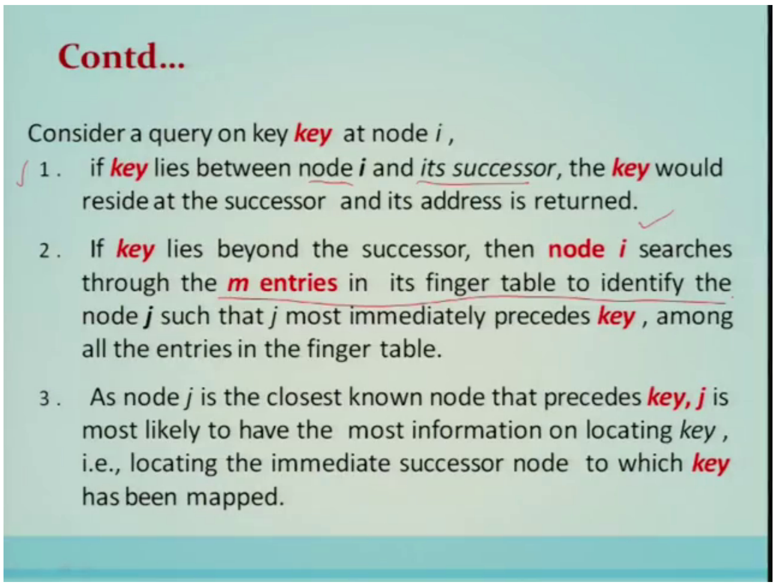
So, due to the log structure of the finger table there is more information about the nodes which are closer than about the nodes which are further away.

So that means, if there are m values are stored in the finger table of a node. So, finger table is also called as a routing table. So, it is saying that the nodes that the entries are more for the nodes which are closer compared to the nodes which are further away are having very little entries and this is called logarithmic structure of the table organization.

So, if the node is very close closely located in the close successors then basically it can be identified in the table itself, but if it is further then it will be having a link to the another table where it becomes a closer to some other node and it will be identified in a very few number of hops and this structure is called logarithmic structure. Now consider a query on a key, key at a node i if the key lies between i and its successor the key would reside at the successor and its address will be returned as you have seen in the simple lookup.

Now, if the key lies beyond the successor then node i searches through m entries in its finger table here it differs from simple lookup because it has more possibilities because it has m different successors in stored in the finger table.

(Refer Slide Time: 34:15)



Contd...

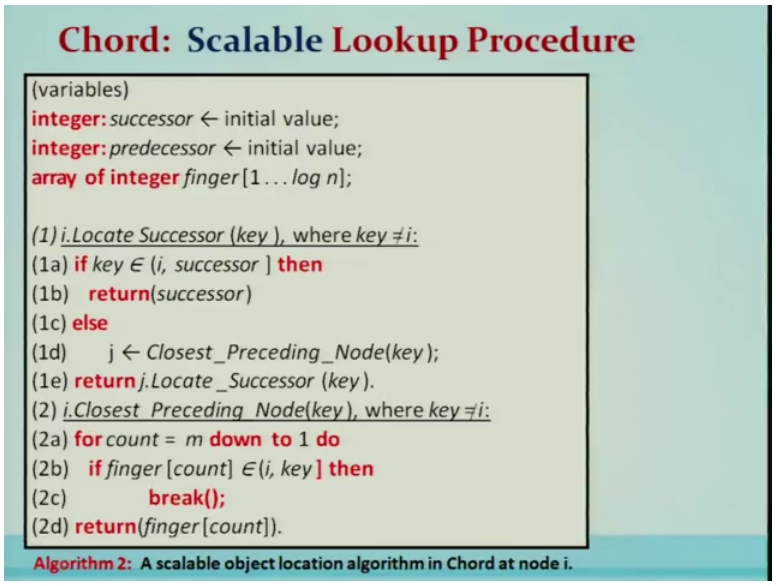
Consider a query on key ***key*** at node *i*,

1. if ***key*** lies between node ***i*** and *its successor*, the ***key*** would reside at the successor and its address is returned.
2. If ***key*** lies beyond the successor, then **node *i*** searches through the ***m* entries** in its finger table to identify the node ***j*** such that *j* most immediately precedes ***key***, among all the entries in the finger table.
3. As node *j* is the closest known node that precedes ***key, j*** is most likely to have the most information on locating *key*, i.e., locating the immediate successor node to which ***key*** has been mapped.

So, this particular m entries will be searched to identify the node j such that j is most immediately preceding that particular key k, among all the entries in the finger table.

Now, where j is the closest known node that precedes key j, so j is the most likely to have a most information on locating the key locate locating the immediate successor node to which the key has mapped the this particular is scalable lookup procedure you can see in this particular algorithm.

(Refer Slide Time: 35:02)



Chord: Scalable Lookup Procedure

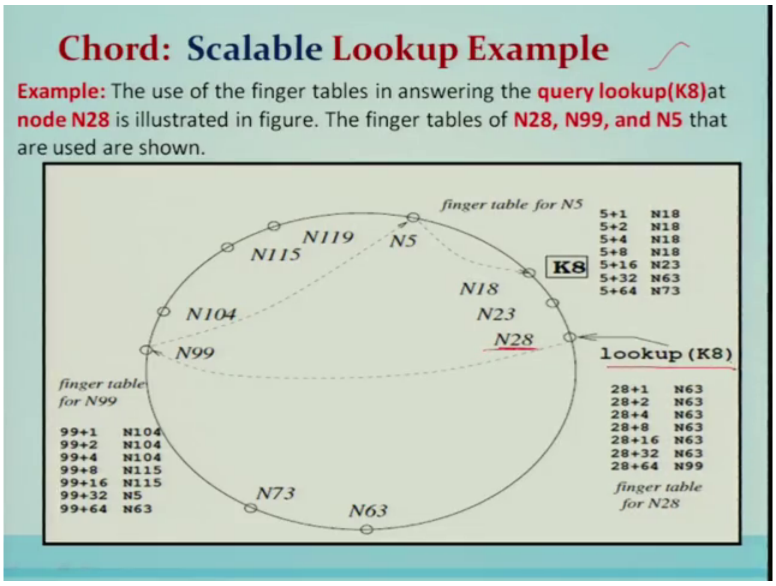
```
(variables)
integer: successor ← initial value;
integer: predecessor ← initial value;
array of integer finger[1 . . . log n];

(1) i.Locate Successor (key ), where key ≠ i:
(1a) if key ∈ (i, successor ] then
(1b)   return(successor)
(1c) else
(1d)       j ← Closest_Preceding_Node(key );
(1e) return j.Locate _Successor (key ).
(2) i.Closest_Preceding_Node(key ), where key ≠ i:
(2a) for count = m down to 1 do
(2b)   if finger [count] ∈ (i, key ] then
(2c)             break();
(2d) return(finger [count]).
```

Algorithm 2: A scalable object location algorithm in Chord at node i.

So, here this algorithm says that if the key lies between i and the successor then the successor will be returned this is same as the simple lookup else what it does it will try to find out this closest preceding node and this closest preceding node procedure goes like this for that particular node it will search through the finger table of m entries.

So, if the node is too far then the last entry will basically be not be able to satisfy the key so; that means, key between i and successor the key lies beyond the successor. So, the last entry of the successor will be returned then in that case because key is lying after this particular last entry in the successor here. So, that way it will be returned and this particular lookup will lead to the new entry. So, let us take the example of scalable lookup let us take that the query of a lookup at K 8 at node 28.

(Refer Slide Time: 36:29)



Chord: Scalable Lookup Example

Example: The use of the finger tables in answering the **query lookup(K8)**at **node N28** is illustrated in figure. The finger tables of **N28, N99, and N5** that are used are shown.

finger table for N5

5+1	N18
5+2	N18
5+4	N18
5+8	N18
5+16	N23
5+32	N63
5+64	N73

finger table for N99

99+1	N104
99+2	N104
99+4	N104
99+8	N115
99+16	N115
99+32	N5
99+64	N63

28+1	N63
28+2	N63
28+4	N63
28+8	N63
28+16	N63
28+32	N63
28+64	N99

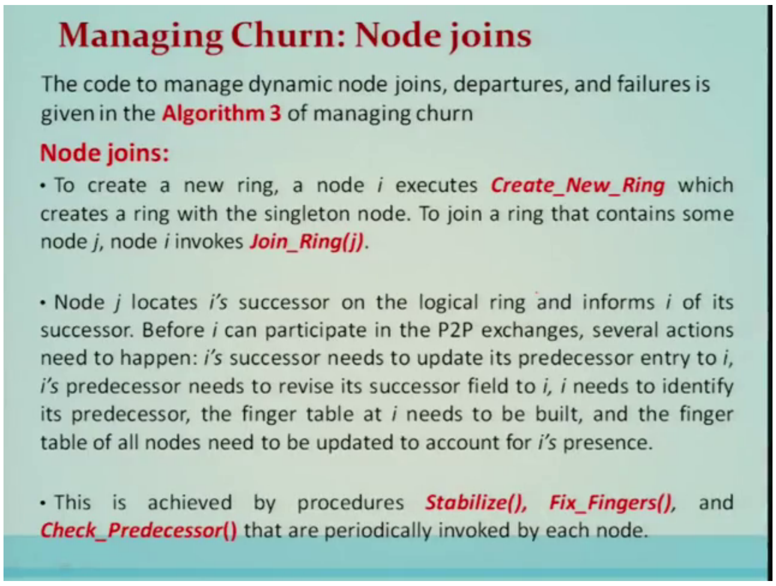
finger table for N28

So, at node 28 there is a lookup of for a key 8 let us see how in scalable lookup. The number of hops are very little compared to the previous simple lookup where too many number of hops were there to locate or to lookup the key value K 8. So, let us see that i is 28 over here and the successor is 63 the key which you are now searching is 8, now 8 does not lie between i and successor.

So, hence we are going to see the next successor up to this point the key does not lies. Then we have to see the last entry that is 99, between 28 and 99 also the key 8 does not lies. So, it lies beyond this, so this particular node will be now looked up. So, its finger table of 99 will be looked up now between i is 99 and the first successor is 104 the key value is beyond that it is not lying. So, basically, it will come to this particular entry where it is N 5.

So, between 99 and N 5 when it comes to 63 it is far away, so N 5 knows where this key 8 is stored. So, the search will go to N 5 and N 5 you just see that N 18 is the node which which is stores K 8. So, 1 2 3 in 3 hops it reaches to K 8 why because see here it refers to the last entry why because in a logarithmic manner this particular lookup table is a stored why because the entry is too far. So, the last entry basically will carry to N 99 furthest node and that node in turn will basically figure out N 5 which contains this information where this ns lookup table or ns finger table it knows this information.

(Refer Slide Time: 39:02)



Managing Churn: Node joins

The code to manage dynamic node joins, departures, and failures is given in the **Algorithm 3** of managing churn

Node joins:

- To create a new ring, a node *i* executes ***Create_New_Ring*** which creates a ring with the singleton node. To join a ring that contains some node *j*, node *i* invokes ***Join_Ring(j)***.

- Node *j* locates *i's* successor on the logical ring and informs *i* of its successor. Before *i* can participate in the P2P exchanges, several actions need to happen: *i's* successor needs to update its predecessor entry to *i*, *i's* predecessor needs to revise its successor field to *i*, *i* needs to identify its predecessor, the finger table at *i* needs to be built, and the finger table of all nodes need to be updated to account for *i's* presence.

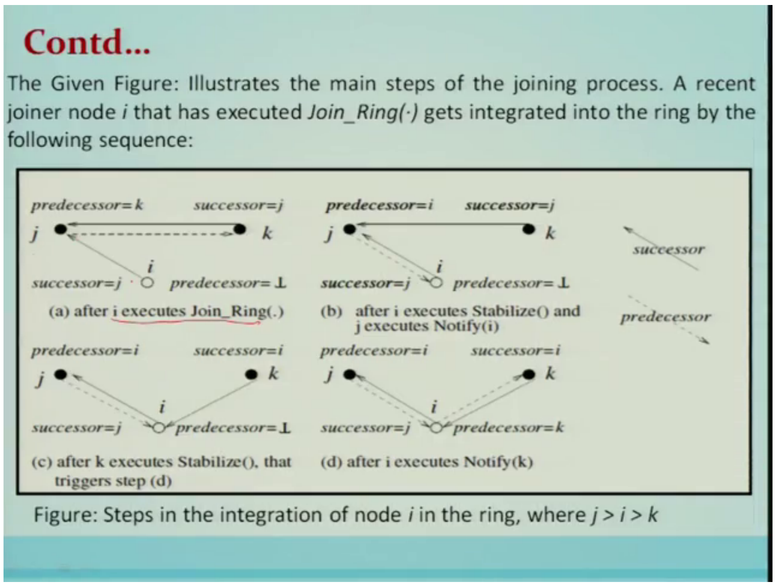
- This is achieved by procedures ***Stabilize()***, ***Fix_Fingers()***, and ***Check_Predecessor()*** that are periodically invoked by each node.

Managing churn that is node joins and node leaves that is called churning. So, the node, the code for managing dynamic node joins departure and failure is given in the next algorithm and that is called managing churn in the chord protocol. Node joins to create a new ring a node i executes create new ring which creates a ring with a singleton node to join the ring that contains some node j node i invokes join ring.

So, this particular joining of a ring is nothing, but just like or inserting a new node in a linked list that kind of scenario will arise over here. So, node j locates i successor. For example, if I wants to join or a new joiner if i is the new joiner, it will locate its successor and let us say that it is j. So, j is successor. So, having located I successor on I the logical ring I will inform of its successor. So, successor will change who is its predecessor from let us say that it is earlier it was K from predecessor value of j which was earlier K has to be changed to i and as far as the K is concerned it will contain the successor of K is equal to let us say j that also has to be changed to i and that way i is inserted, now that is not all.

Once I gets into the ring then now it has to create its own finger table and all other finger table need to be updated and that is all achieved by the procedure called stabilize fixed fingers and check predecessors that are periodically invoked by each node. So, all that things whatever I have explained in the previous slide is shown in this particular figure. So, here when I execute a join ring here, it tries to enter into the a position into a ring in a form of an id and, I will identify its successor let us say the successor is j. So, the predecessor which is earlier K has to be removed you just see that it is changed to i.

(Refer Slide Time: 41:22)



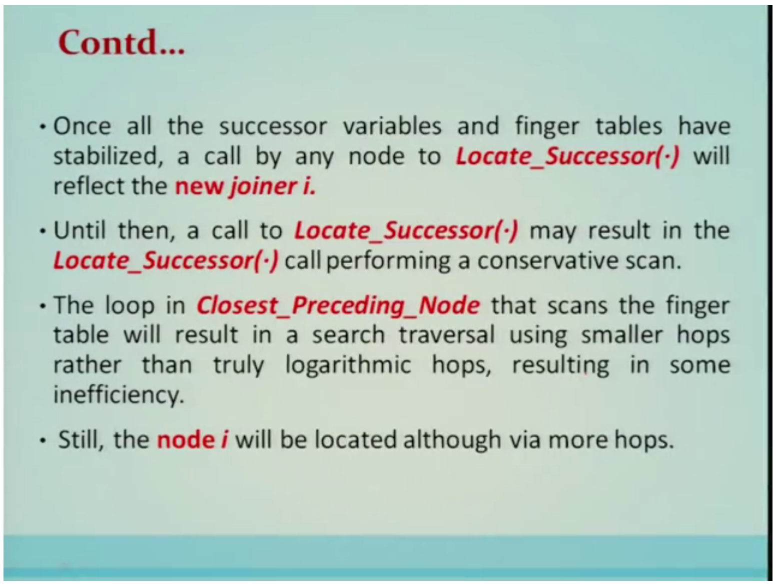
Contd...

The Given Figure: Illustrates the main steps of the joining process. A recent joiner node *i* that has executed *Join_Ring(·)* gets integrated into the ring by the following sequence:

Figure: Steps in the integration of node *i* in the ring, where *j* > *i* > *k*

Similarly, the i is successors successor which is basically earlier positioning as particular j now has to be changed in this particular manner as particular i and that is called predecessor, predecessor of i has to be changed. So, this is a node insertion quite simple. So, once all the successor variables and a finger table have been stabilized a call by any node to locate successor will reflect the new joiner i, until then a call to locate successor may result in a locate successor call performing the conservative scan.

(Refer Slide Time: 42:10)



Contd...

- Once all the successor variables and finger tables have stabilized, a call by any node to *Locate_Successor(·)* will reflect the **new** ***joiner i***.
- Until then, a call to *Locate_Successor(·)* may result in the *Locate_Successor(·)* call performing a conservative scan.
- The loop in *Closest_Preceding_Node* that scans the finger table will result in a search traversal using smaller hops rather than truly logarithmic hops, resulting in some inefficiency.
- Still, the **node** ***i*** will be located although via more hops.

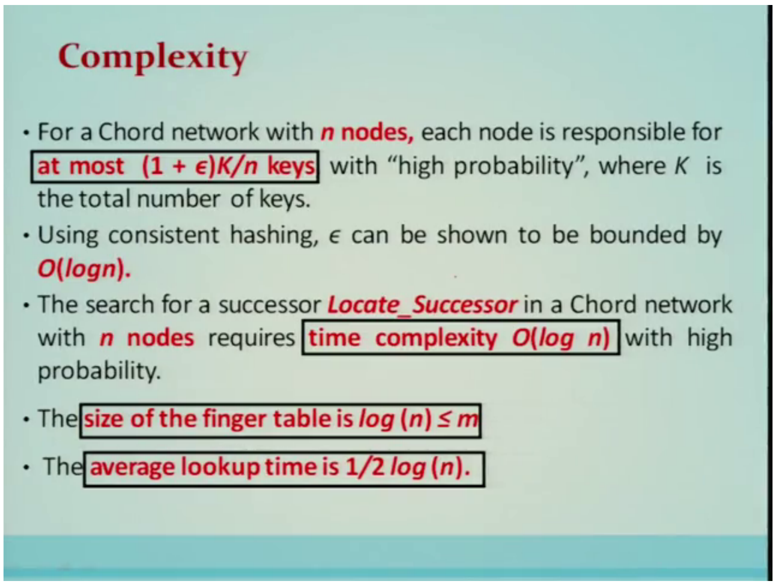
The loop in the closest preceding node that is scans the finger table will result in a search traversal using a smaller number of hops rather than truly logarithmic hops resulting in some inefficiency.

So, still the node i will be located through via more hops. So, it says that till the finger table is being built it will perform the simple lookup, hence it is not the logarithmic number of hops it will be basically linear number of hops that is of the order n, but after sometimes it stabilizes and populates its finger table then basically it gets the benefits.

Now, managing churn the node failures and departures now when a node fails then check predecessor which periodically checks will i be identified that that the node is not working or a fail and hence basically these entries are to be modified. So, node i gets a churn to update the predecessor field when another node K causes i to execute notify K because, but that can happen only if Ks successor variable is i. This requires the predecessor of the failed node to recognize that it successor has failed and get a new functioning successor. So, where a node fails using these check predecessor and notify will take care that another node is in place; that means, successor and predecessor values are adjusted. So, there the new predecessor a new successor comes into the effect.

Note that from algorithm 3 that knowing that a successor is functional and the nodes pointed to by the finger pointers are functional is essential. This is all is mentioned in this particular code now we are talking about the complexity for chord network with n nodes each node is responsible for at most one plus epsilon K upon n keys with high probability where K is the total number of keys. Using consistent hashing epsilon can be shown to be bounded by an order log n the search for successor locate successor in chord with n requires the time complexity of the order log n with high probability.

(Refer Slide Time: 44:40)

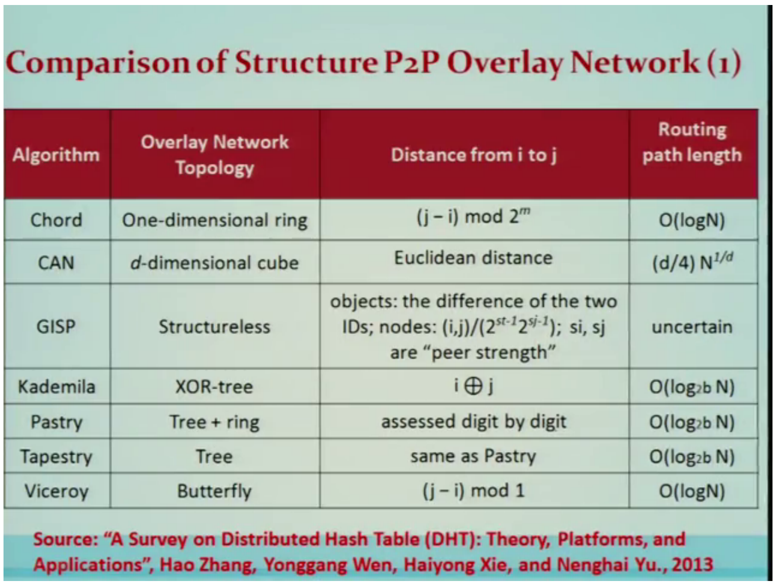


Complexity

- For a Chord network with ***n* nodes,** each node is responsible for **at most $(1+\epsilon)K/n$ keys** with "high probability", where K is the total number of keys.
- Using consistent hashing, ϵ can be shown to be bounded by ***O(logn).***
- The search for a successor ***Locate_Successor*** in a Chord network with ***n* nodes** requires **time complexity *O(log n)*** with high probability.
- The **size of the finger table is *log (n)* ≤ *m***
- The **average lookup time is 1/2 *log (n)*.**

The size of the finger table is log of n is bounded by m and the average lookup time is one upon 2 log n.

(Refer Slide Time: 44:55)



Comparison of Structure P2P Overlay Network (1)

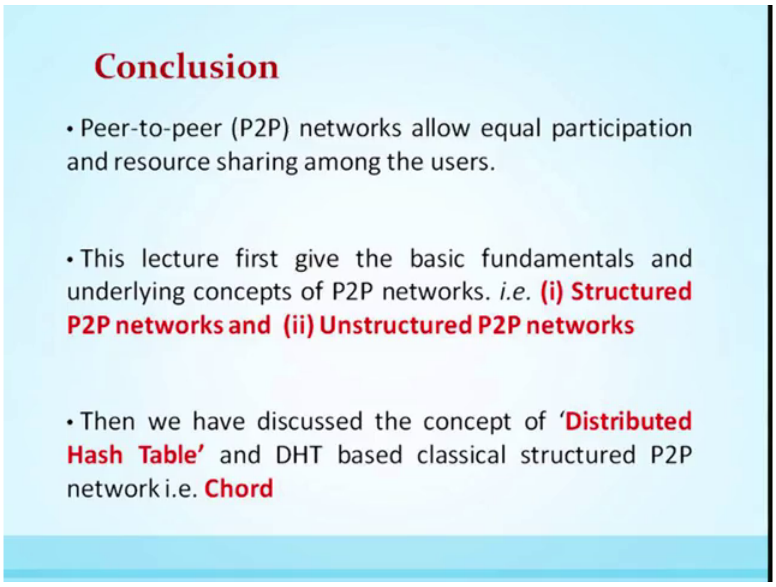
Algorithm	Overlay Network Topology	Distance from i to j	Routing path length
Chord	One-dimensional ring	(j – i) mod 2^m	O(logN)
CAN	*d*-dimensional cube	Euclidean distance	(d/4) $N^{1/d}$
GISP	Structureless	objects: the difference of the two IDs; nodes: (i,j)/($2^{st-1}2^{sj-1}$); si, sj are "peer strength"	uncertain
Kademila	XOR-tree	i ⊕ j	O($\log_{2b}$ N)
Pastry	Tree + ring	assessed digit by digit	O($\log_{2b}$ N)
Tapestry	Tree	same as Pastry	O($\log_{2b}$ N)
Viceroy	Butterfly	(j – i) mod 1	O(logN)

Source: "A Survey on Distributed Hash Table (DHT): Theory, Platforms, and Applications", Hao Zhang, Yonggang Wen, Haiyong Xie, and Nenghai Yu., 2013

So, comparison of structure peer-to-peer network, so we have discussed the chord protocol similarly the can GISP, Kademila, Pastry, Tapestry, Viceroy is the another form which have different structure to basically design the overlays and that is why these different routing path lengths are being obtained.

Comparison of structure peer-to-peer overlay if we see that the routing path is a greedy algorithm and nodes joining and nodes leaving all these are being supported.

(Refer Slide Time: 45:41)



Conclusion

- Peer-to-peer (P2P) networks allow equal participation and resource sharing among the users.
- This lecture first give the basic fundamentals and underlying concepts of P2P networks. *i.e.* **(i) Structured P2P networks and (ii) Unstructured P2P networks**
- Then we have discussed the concept of **'Distributed Hash Table'** and DHT based classical structured P2P network i.e. **Chord**

So, conclusion peer-to-peer networks allow equal participation and resource sharing among the users. This lecture first gives the basic fundamentals of underlying principles of peer-to-peer network that is the structured peer-to-peer network and unstructured peer-to-peer network. Then we have discussed the concept of distributed hash table using the concept of consistent hashing and distributed hash table based classical structured peer-to-peer network that is in the form of logical ring structure we have seen in the chord protocol.

Thank you.