Distributed Data Management
Part 3 - Peer-2-Peer Systems (cont)

Overview

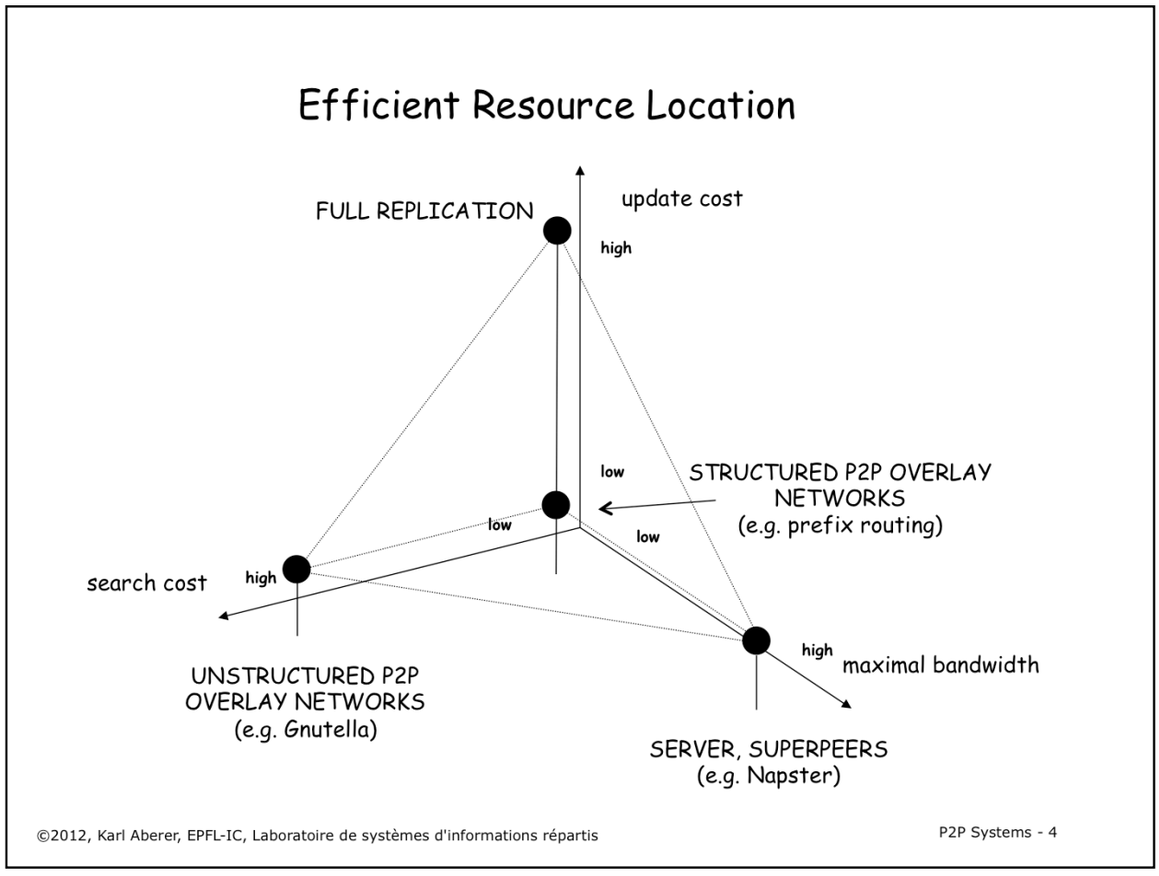
1. P2P Systems and Resource Location
2. Unstructured P2P Overlay Networks
3. Hierarchical P2P Overlay Networks
4. Structured P2P Overlay Networks
5. Small World Graphs
6. P2P Data Management

4. Structured P2P Overlay Networks

- Unstructured overlay networks – what we learned
  - simplicity (simple protocol)
  - robustness (almost impossible to "kill" – no central authority)
- Performance
  - search latency O(log n), n number of peers
  - update and maintenance cost low
- Drawbacks
  - high bandwidth consumption for search
  - free riding

- Can we do better?

Unstructured overlay networks have been successful in practical applications, due to their simplicity in design and relative efficiency. However, a major drawback is that the exhaustive search mechanisms leads to a heavy network usage. It has been shown that at certain times P2P traffic has been the most dominant one on the Internet, which has also lead to concerns by the network operators. This issue has incited efforts to develop alternative approaches for performing search in peer-to-peer networks, which are summarized under the term structured overlay networks.



This figure illustrates the design space for overlay networks in terms of three critical performance parameters:

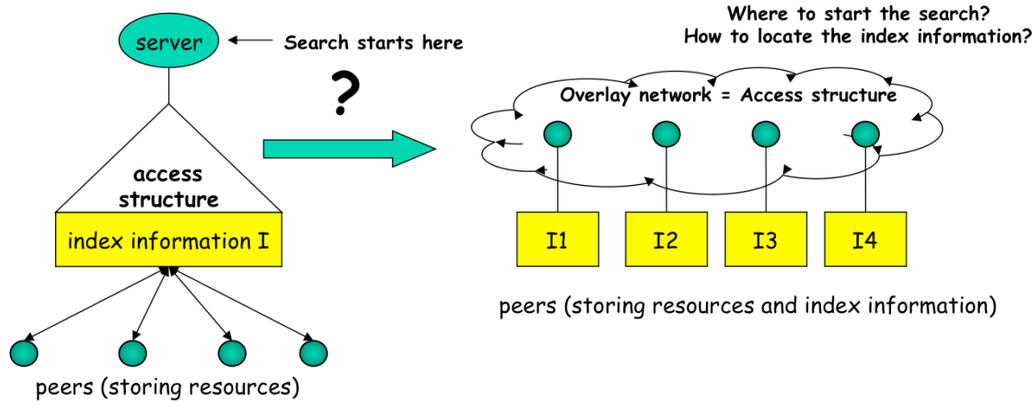
1. Bandwidth consumption for search
2. Bandwidth consumption for updates
3. Maximal bandwidth usage at a single node

We see that both unstructured overlay networks and superpeer or server architectures, suffer performance along one of the dimensions. We could avoid these performance problems by fully replicating data in the overlay network. In terms of available storage this solution would be perfectly feasible for many peer-to-peer applications. However, it would suffer from high update cost.

In the following we will introduce an alternative approach that indeed aims at optimizing all three of these performance dimensions simultaneously.

Structured P2P Overlay Networks

- Goal: efficient search using few messages without designated servers
- Easy: distribution of index information over all peers
  - every peer maintains and provides part of the index information (k, p)
- Difficult: distributing the access structure to support efficient search
  - Realized by an "overlay network"

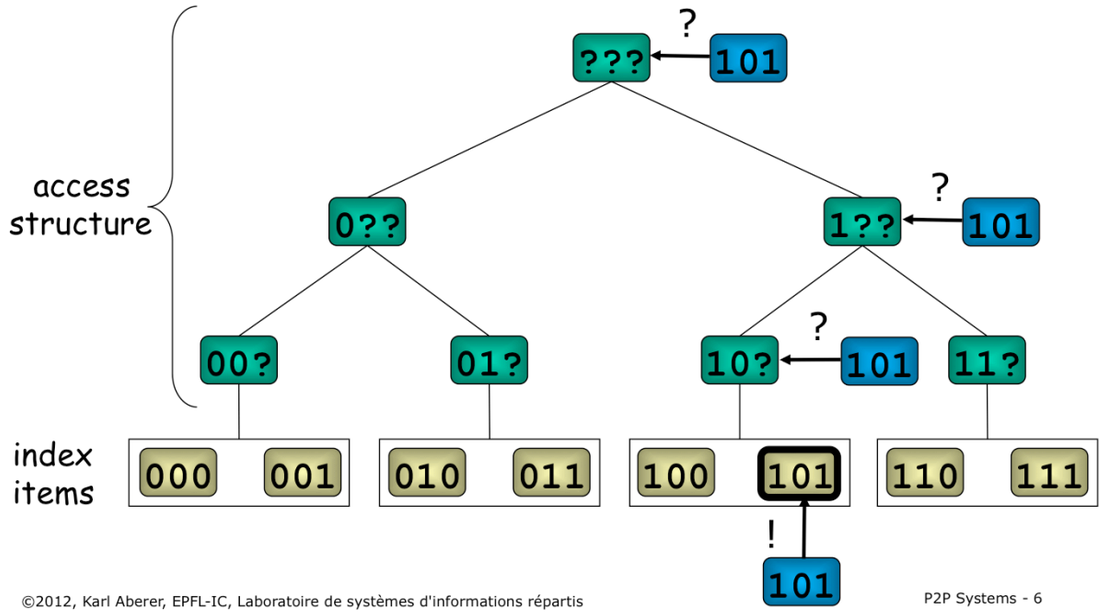


Hierarchical P2P networks introduce multiple levels at which nodes take over different roles, in order to support efficient access to the data. One of the reasons why the index information is concentrated on few nodes, is the possibility to construct at these nodes a data access structure in order to efficiently locate an index item. (We have to be here careful with the term "index" – in the P2P area it is used to designated the relation that binds peer addresses to data keys. However, in the area of data management an index is usually a data structure, such as a tree or a hash table, that supports efficient access to a set of keys and the data associated with that key. For clarity we will call this data structure in the following the data access structure or simply access structure).

It is fairly straightforward to distribute the index information over the peers, by partitioning it horizontally, but what can be done about the data access structure ?

Example: Scalable Distributed Tries (P-Grid)

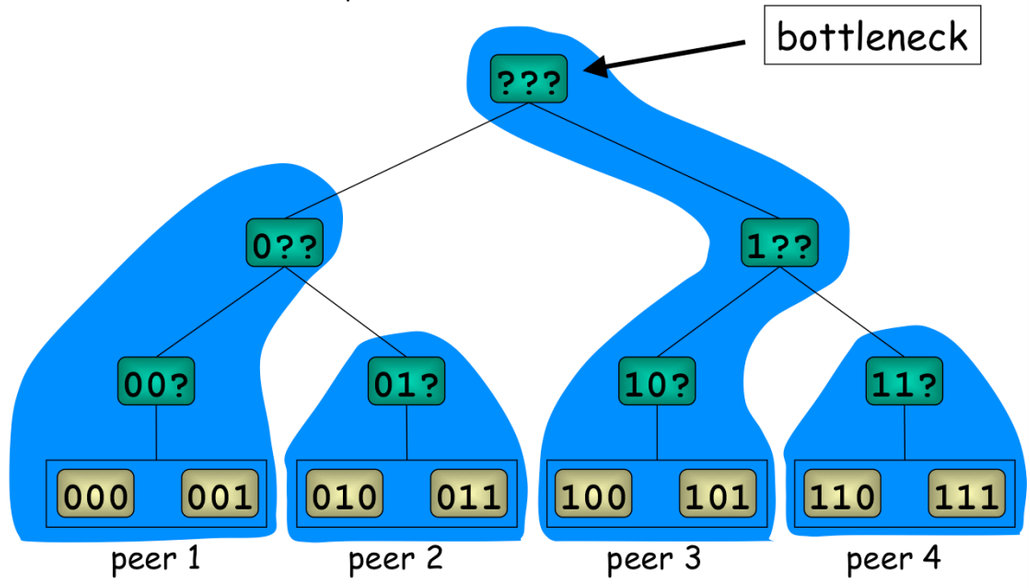
- Search trie: search keys are binary keys



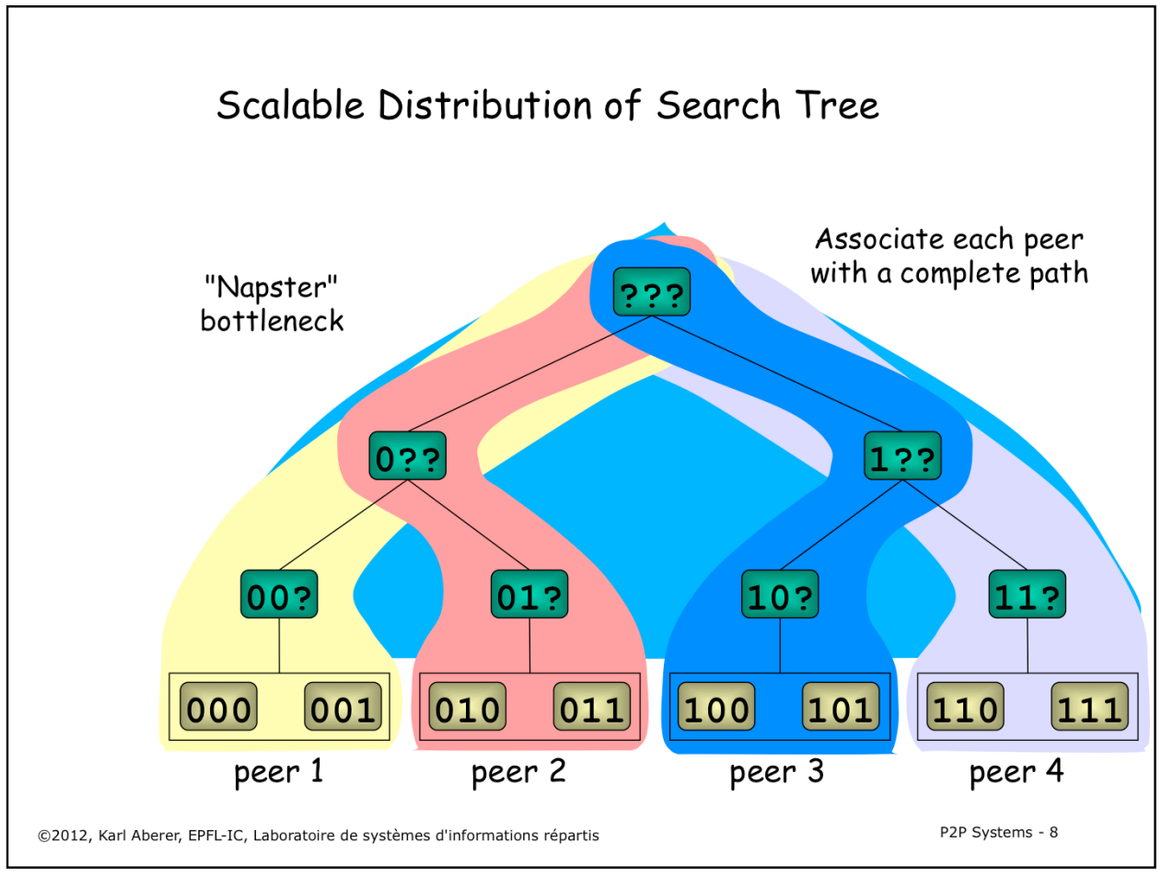
Our first example of distributing a data access structure is based on the idea to distribute a search tree in a scalable manner. This idea is employed by a number of different approaches, including Kademlia which is now used in as search mechanism in popular file-sharing systems, but also P-Grid and Pastry. We will base our description on P-Grid. In the following we will more precisely consider binary search tries as the tree-based search structure.

Non-scalable Distribution of Search Tree

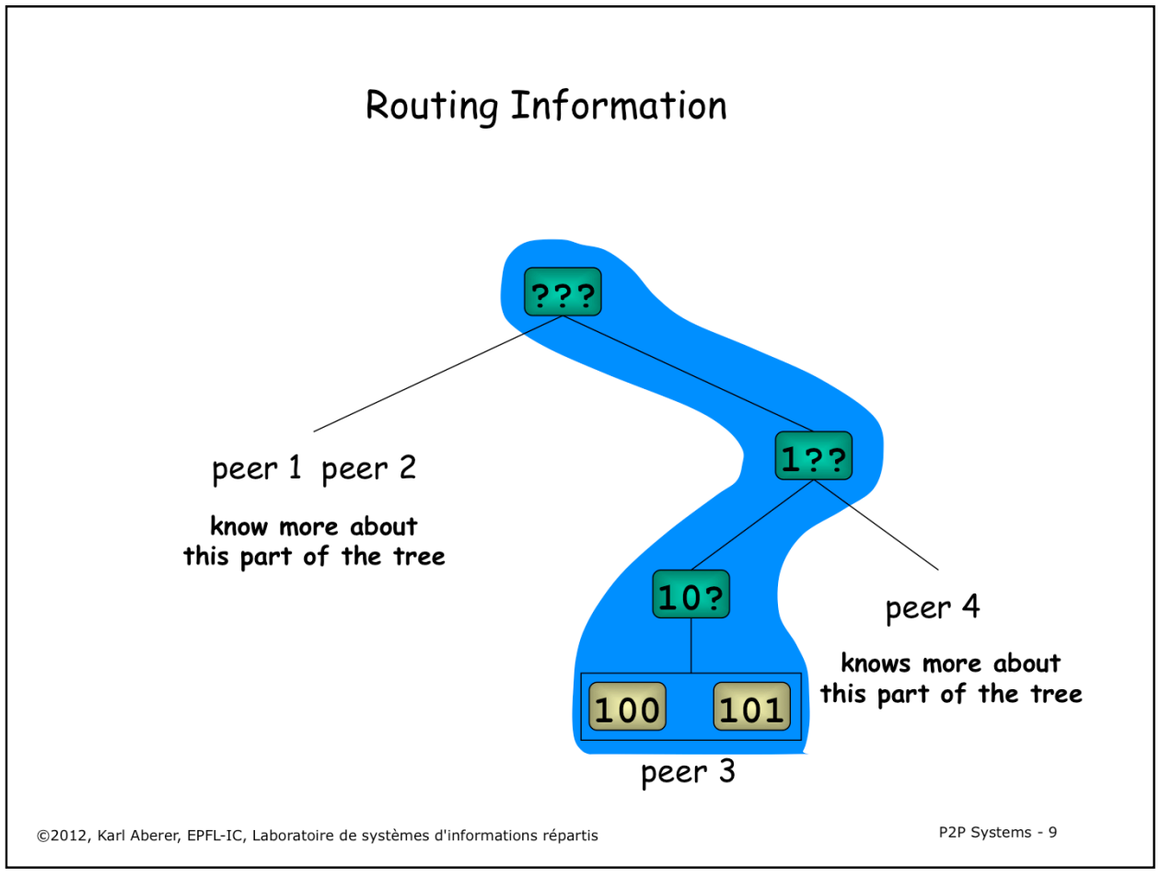
- Distribute search tree over peers



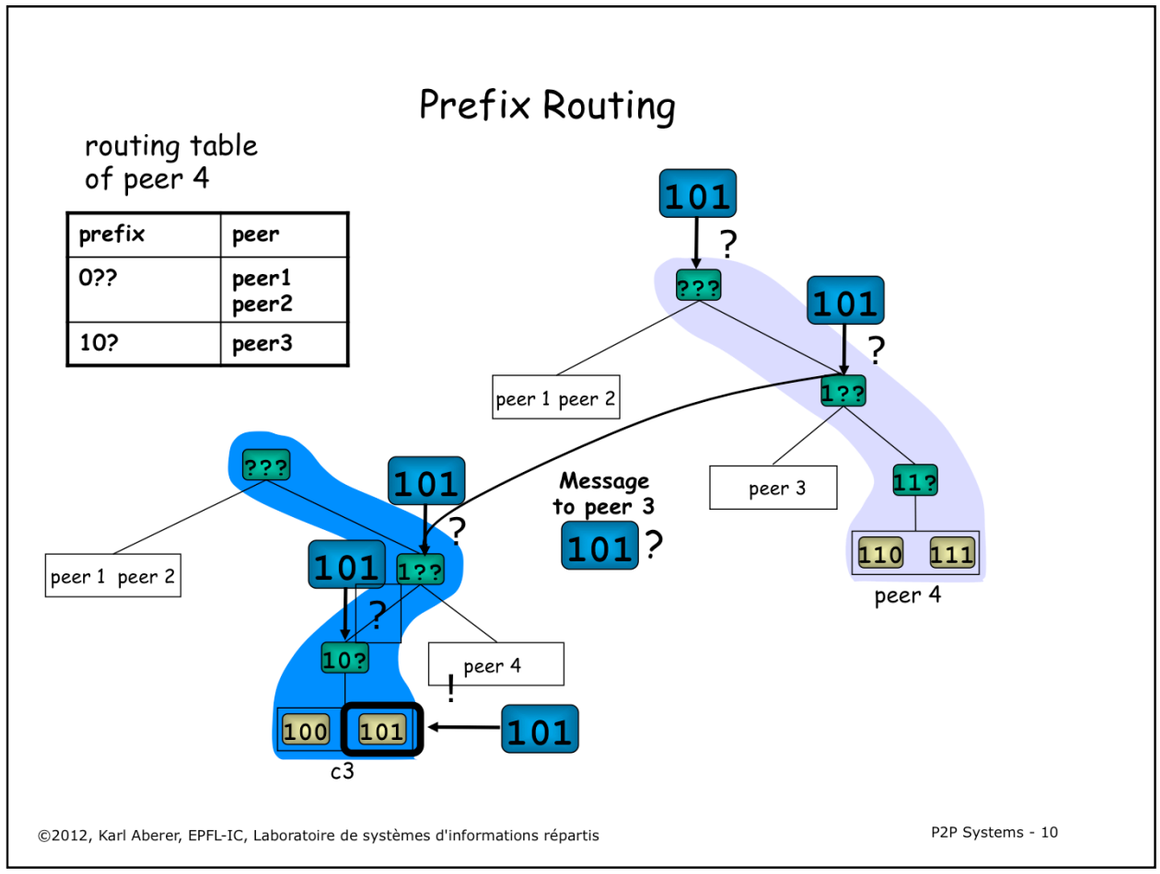
In order to make the search tree available in a distributed environment the search tree needs to be distributed over the peers that hold the index items that are accessed through the search tree. One possibility for doing this is indicated above: peers partition the data space and hold the corresponding index items. In addition they store a part of the search tree that belongs to the path from their leaf to the root. Storing part of a search tree means that the peer know the locations (Internet addresses) of the peers that this part of the search tree is connected to. Partitioning the search tree in the manner shown in the illustration (which in fact is an approach for distributing data access structures in cluster computing) leads to a bottleneck and single point of failure at the peer that holds the root of the tree.



Another possibility is to store the whole tree at one peer: this is the organization hierarchical overlay networks use. A better idea is the following: each peer stores a copy of all the nodes of the tree that lead from the root to its own data. This results in copying nodes at higher levels of the tree to multiple peers. For example, each peer holds a copy of the root node. This is in fact good, because it allows to start searches in the tree from every peer and thus bottlenecks are avoided. The replication of tree nodes in this way does not incur substantial storage cost, since the paths of the tree are of length O(log n) and thus the required additional storage space is also O(log n).

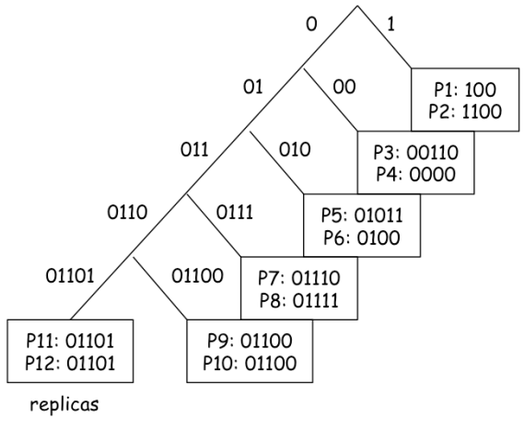


From the perspective of a single peer (e.g. peer 3), now the network appears as follows: the peer knows for each level of the search tree one continuation in the search tree, and for the alternative paths it knows about some other peers in the network, that hold information on that path.



The resulting distributed search structure is called a P-Grid. A search can start at any peer, because every peer has a copy of the root node. For example, consider a search at peer 4 for 101. Since peer 4 holds the tree node needed for processing requests starting with 1 it can use this node to traverse one level down in the tree. At this point the node for processing queries starting with 10 is missing. But peer 4 knows from it's routing table that peer 3 holds such a tree node. Therefore it sends the request to peer 3. After peer 3 receives the messages it can successfully answer the request.

P-Grid Routing Tables and Search



Example: routing table of a peer with path 01101

Peer with path $p = p_1,..,p_l$,$p_i \in \{0,1\}$, stores routing table:

- For prefix $p_1,..,p_j$, j=1,..,l a constant number r of references to peers with path $p_1,..,1\text{-}p_j$
- A constant number r of references to replicas with the same path

```
search(p, k)
if
  k=path(p) then return(p) //found
else
  find in routing table peer_i with
  longest prefix matching k
  search(peer_i, k)
```

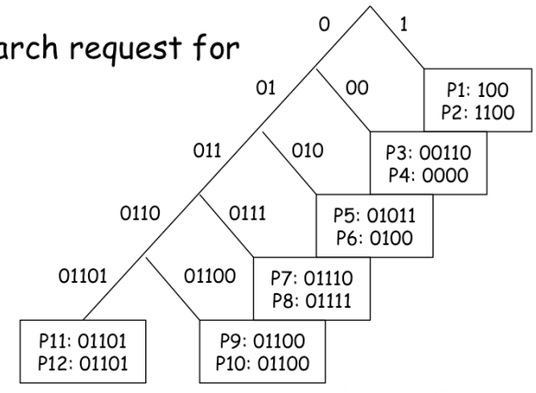
Search cost bound by routing table size: $\log_2(n)$ for balanced tree

For better resilience each peer maintains several redundant references to peers in the other parts of the search tree, as well as a list of replicas of peers storing the same path. These entries for the routing table of the peer are obtained through interactions with other peers, similarly to the ping-pong mechanism that is used by Gnutella. There exist a number of variations of such maintenance schemes, but we will not further describe them here.

Questions

- The index information in a structured overlay network
  1. Provides references to route a search request within the overlay network
  2. Provides for a given key the reference to the peer that stores the resource
  3. Is replicated in routing tables to support redundant search paths

- For the given routing table, the search request for the key 0101 is routed
  1. Always to peer P5
  2. Either to peer P5 or P6
  3. Either to peer P3, P4, P5 or P6



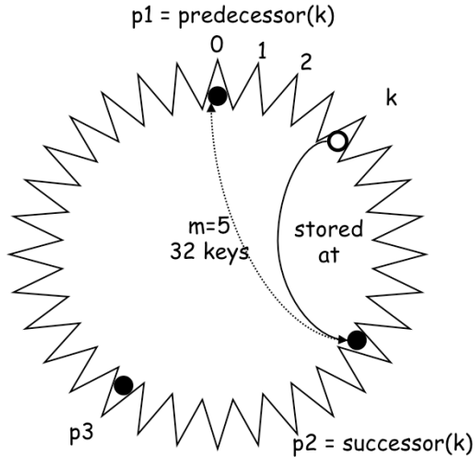
Structured P2P Overlay Network Approaches

- Different strategies
  - P-Grid: distributing a binary search tree
  - Chord: constructing a distributed hash table
  - CAN: Routing in a d-dimensional space
  - FreeNet: caching index information along search paths
- Commonalities
  - each peer maintains a small part of the index information
  - each peer maintains a small routing table for routing in the overlay network
  - searches are performed by directed message forwarding
- Differences
  - performance and qualitative criteria

To solve the problem of distributing a data access structure a variety of approaches have been developed, that all try to achieve the same goal, namely performing searches not only with low latency but also by consuming only little network bandwidth. We have already seen one possible approach, and In the following we will study 3 other approaches, each representative for a different paradigm and for a class of related approaches.

Example 2: Distributed Hash Tables (Chord)

- Hashing of search keys AND peer addresses on binary keys of length **m**
  - e.g. m=5, key("jingle-bells.mp3")=4, key(196.178.0.1)=19
- Data keys are stored at peer with next larger peer key



peer with hashed identifier p,
data with hashed identifier k,
if k ∈] predecessor(p), p]
then k stored at p

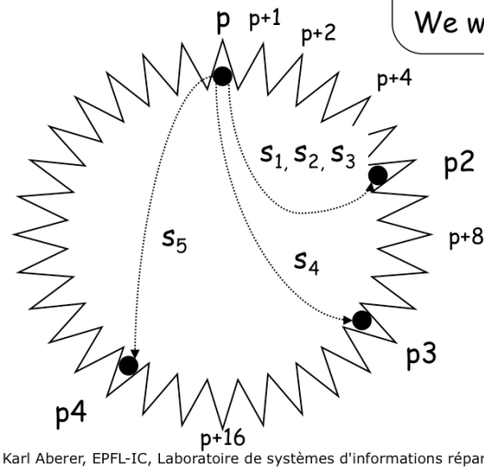
Search strategies
1. every peer knows all others
   O(n) routing table size
2. peers know successor only
   O(n) search cost

Chord is based on the idea of distributing a hash table (instead of a search tree). First, data and node identifiers are mapped into the same identifier space using a uniform random hash function. We assume that the identifiers (or keys) are arranged on a circle (or in other words all computations on hashed values are performed modulo m). Then nodes become responsible for storing the data that belongs to "their" interval, which is defined as all key values preceding the node. Also all nodes store a reference to their successor, such that the nodes form a ring topology. This ring can be used for search, however, the search cost would be linear in the size of the network. Alternatively all peers could learn about all other peers, but then routing tables would grow linearly in the size of the network. Chord provides a solution to keep both search cost and routing table size scalable.

Chord Routing Tables

- Idea: every peer knows m peers at exponentially increasing distance

Peer p stores its successor(p) and the first peer with hashed identifier s_i such that s_i =successor(p+2^{i-1}) for i=1,..,m
We write also s_i = finger(i, p)



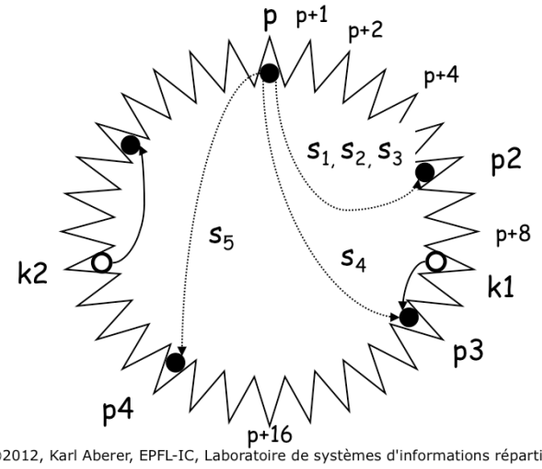
Routing table size: m

i	s_i
1	p2
2	p2
3	p2
4	p3
5	p4

In order to provide efficient search, i.e. in O(log n) time, with acceptable storage overhead (i.e. O(log n) space) routing tables are constructed. They are designed such that peers know other peers in intervals of exponentially increasing size. That means, for key values close to the peer, the peers know other peers at a finer granularity, whereas for key values which are further away, the distances between known peers increase. Since the interval lengths increase exponentially, the routing tables are logarithmic in the size of the network.

Search in Chord

```
search(p, k)
find in routing table largest (i, p*) such that p* ∈ [p,k[
if such a p* exists then search(p*, k)
else return (successor(p)) // found
```



Expected search cost:
$O(\log n)$

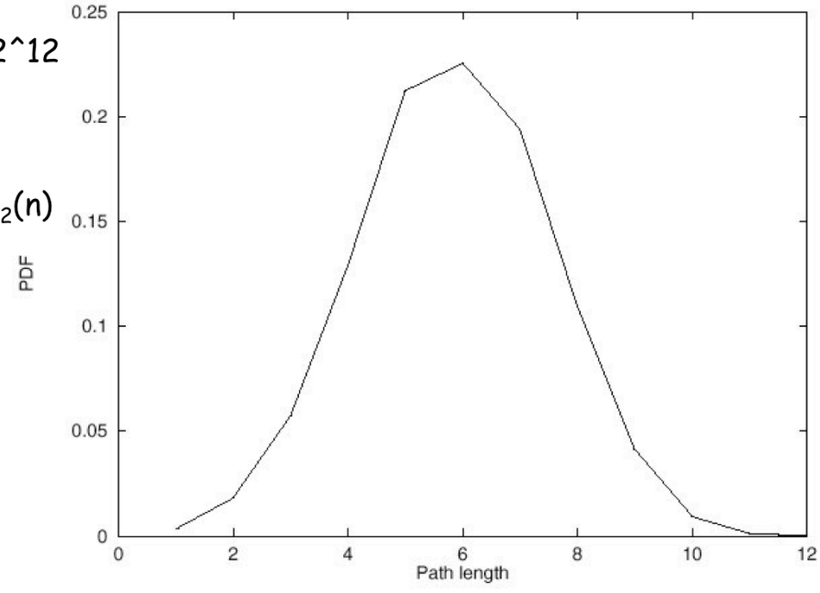
Search proceeds now in the obvious way. When a search request arrives at a peer, it finds in its routing table the largest peer key that is smaller than the searched data key. Now there exist two possibilities: either there exists no such peer, then the peer knows its successor is responsible for the key, and it returns its address as the search result, or there exists such a peer and then the request is forwarded to the successor peer. Since the routing table entries are at exponentially increasing distances, it can be shown that the search can be performed in logarithmic time (with high probability).

Length of Search Paths

Network size n=2^12

100 2^12 keys

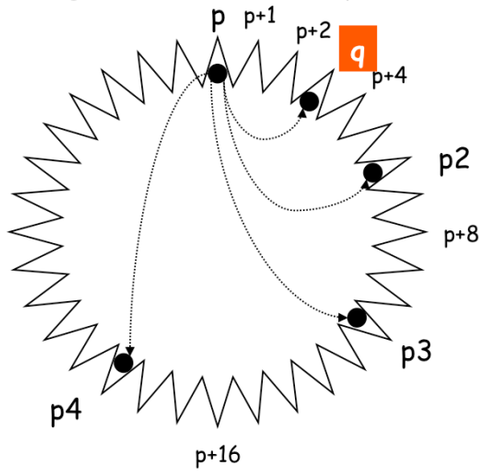
Path length $\frac{1}{2}$ $\log_2(n)$



Experimental evaluations show that the search performance in Chord is as expected very good. The length of the search paths is closely concentrated around ½ $\text{Log}_2(n)$. The factor ½ is explained by the fact that the location where the search starts, is on average at distance ½ (of the ring) to the target.

Maintenance of Chord

- Maintain the integrity of routing tables if peers join or leave
- Example Chord: New node q joining the network
  - Successor nodes need to be updated: successor(p) = q, successor(q) = p2
  - Finger tables need to be updated: both at new and existing peers



routing table of p

i	s_i
1	q
2	q
3	p2
4	p3
5	p4

routing table of q

i	s_i
1	p2
2	p2
3	p3
4	p3
5	p4

Expected cost: O(log^2 n)

So far we have not considered network dynamics for structured overlay networks, i.e. joining and leaving of nodes. This dynamics is called in the literature “churn”. In Gnutella the ping-pong protocols provides the maintenance mechanism to ensure that the network structure is maintained properly. For unstructured networks this problem is relatively easy to solve, since all that needs to be know is a fixed set of (random neighbors). For structured overlay networks the situation is more complex. There a number of structural properties of the overlay network need to be ensured to maintain a consistent network structure. We illustrate this for the example of Chord. When a new node joins it has to ensure the following:

1. Proper connection to its successor node
2. Taking over the keys from it’s successor that it has become responsible for
3. Constructing the routing table

In particular constructing the routing table is somewhat costly. The peer can perform this operation by repeatedly searching for the keys at exponentially increasing distances. Therefore the cost is O(log^2 n). In addition, other peers may be affected by the addition of the new peer and have to update their routing table entries and successor node correspondingly, which can be done at the same cost.

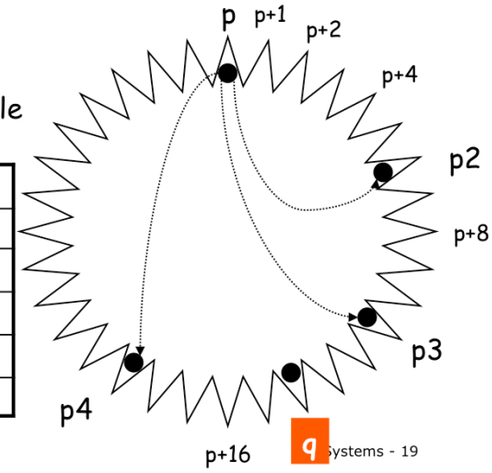
The situation is complicated further when peers enter or leave simultaneously. As changes may then occur in parallel, inconsistencies may be introduced. Even the correct ring connectivity might be compromised. Addressing these problems requires more elaborate maintenance mechanisms, which explains we strongly structured overlay networks such as Chord are preferably used in environments with little churn, whereas in high churn environments loosely structured such as FreeNet that we introduce later(or Kademlia) are more popular.

Questions

- When routing in Chord
  1. The next hop is always uniquely determined
  2. The next hop can be chosen among a constant number of possible candidates
  3. The next hop can be chosen among log n possible candidates
- When adding q to the Chord ring: in the routing table of p
  1. Entries for i=1,2,3,4 change
  2. The entry for i=4 changes
  3. The entry for i=5 changes
  4. No entry changes

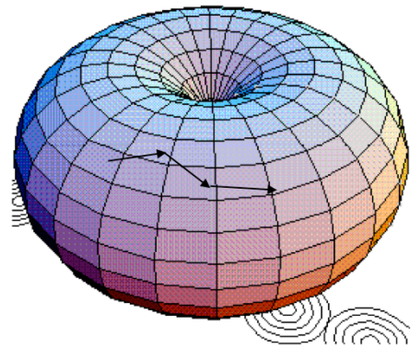
routing table of p

i	s_i
1	p2
2	p2
3	p2
4	p3
5	p4



Example 3: Topological Routing (CAN)

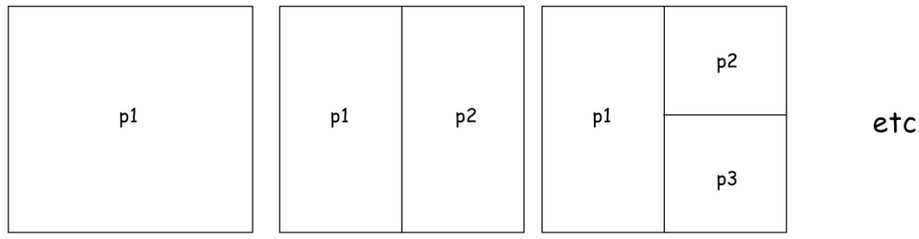
- Based on hashing of keys into a *d-dimensional space* (a torus)
  - Each peer is responsible for keys of a subvolume of the space (a zone)
  - Each peer stores the adresses of peers responsible for the neighboring zones for routing
  - Search requests are greedily forwarded to the peers in the closest zones



Topological routing exploits the fact that in a d-dimensional grid with a fixed number of grid cells (which we imagine are occupied by different peers), the average distance between the cells drops as the dimensionality d grows. The overlay network is thus constructed by assigning both to peer and data keys cells of a d-dimensional grid space, more precisely a d-dimensional torus, where the dimension d is usually not too high (e.g. 2-10). Peers are responsible for cells in the grid, which means they store data items with keys belonging to this volume. Each peer is connected with it's direct neighbors in the grid and search requests are routed to neighboring peers greedily. This means that for performing a search for a key a peer chooses the neighbor for forwarding the request that maximally reduces the distance to the target. This approach is called content-addressable networks = CAN.

CAN Zones

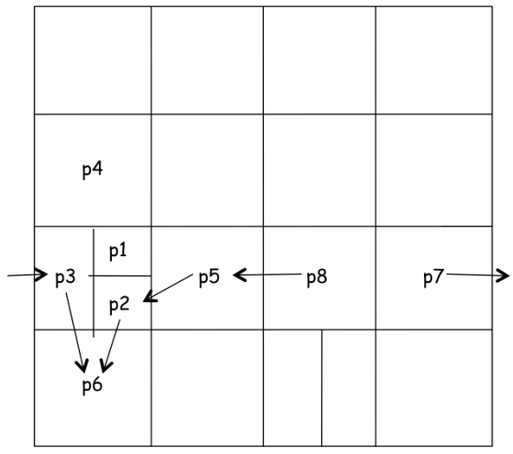
- Example: d=2
  - Space is recursively split along each dimensions as more peers join



  - Peers maintain references to their neighboring zones: neighbors(p1) = {p2,p3}

The zones that peers are associated with in CAN are generated by recursively splitting the space along each of the dimensions in a given order. The peers maintain links to direct neighbors. The example shows how the splitting is performed initially. More details on how the CAN network is constructed and maintained follow later.

CAN Routing Tables and Search



Each peer p stores a routing table with 2d entries containing the two closest 2 neighbors in each dimension

search(p,k)
if p=k then found else
find among neighbors p* with minimal Euclidean distance to k, search(p*,k)

neighbors(p1) = {p2,p3,p4,p5}
neighbors(p3) = {p1,p6,p7,p4}

Example: search starting at p7, p8 for p6

Routing table size:
2d
Expected search cost:
O(d n^(1/d))

This figure illustrates the search algorithm in CAN. A search starting at a zone proceeds greedily to the (or a) next zone which is closer to the target. Note that the space is a torus, e.g. the search starting at p7 for p6 moves to p3.

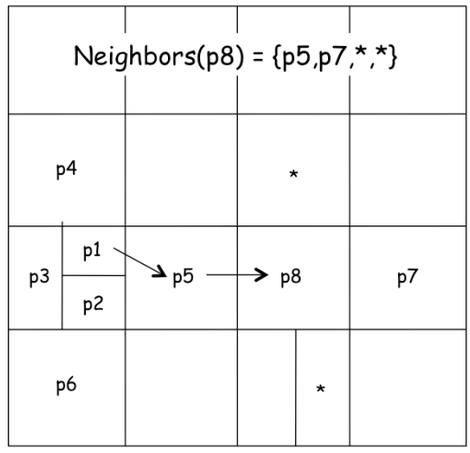
The average search cost using CAN, assuming n peers are uniformly distributed in the grid space of dimension d, is ¼ d n^(1/d). The factor n^(1/d) reflects the reduction in distance due to increasing dimensionality d, the factor d reflects the fact that for performing on step to approach the target in the d-dimensional space (e.g. along the diagonal) d zones have to be traversed, and the factor ¼ corresponds to the fact that the maximal distance among two points is ½ (due to the torus topology) and thus the average initial distance ¼.

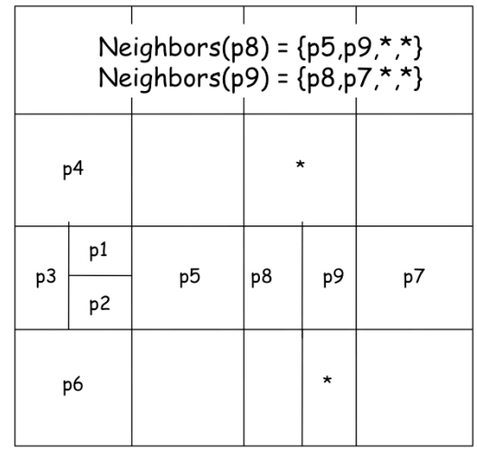
By choosing a proper dimension it would be possible for a fixed size of the network to achieve the same search performance as with tree-based approaches (P-Grid, Chord) of O(log n).

Network Join in CAN

- Node joining the network
  - Choses address (coordinate in d-dim. space)
  - Performs search for the address
  - Splits the region with the node currently managing it
  - Updates to own and neighboring nodes routing tables

Expected cost:
O(d n^(1/d))

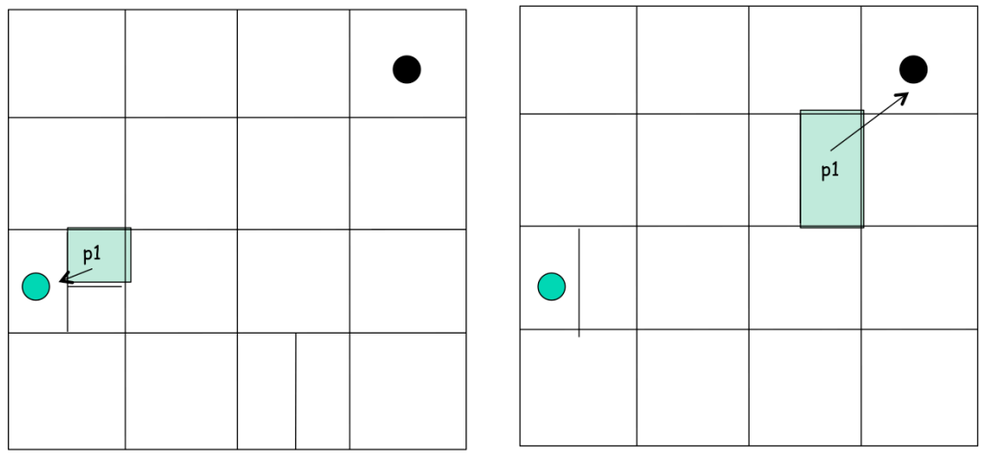




The problem of maintenance under churn is easier to address in CAN, as all changes are localized to the immediate neighborhood of a peer. When a node joins the network it can decide for which zone it would like to be responsible by selecting a coordinate. This is illustrated by the figure. Assume peer 9 decides for a point that lies in the zone that peer 8 is currently responsible for and joins the network at peer 1. First it performs a search for this point finding peer 8, then splits the zone with peer 8. Each of the two nodes are from then responsible for one half of this zone. The routing tables need to be updated for the neighborhoods of peer 8 and peer 9, which requires O(d) operations.

Multiple Realities

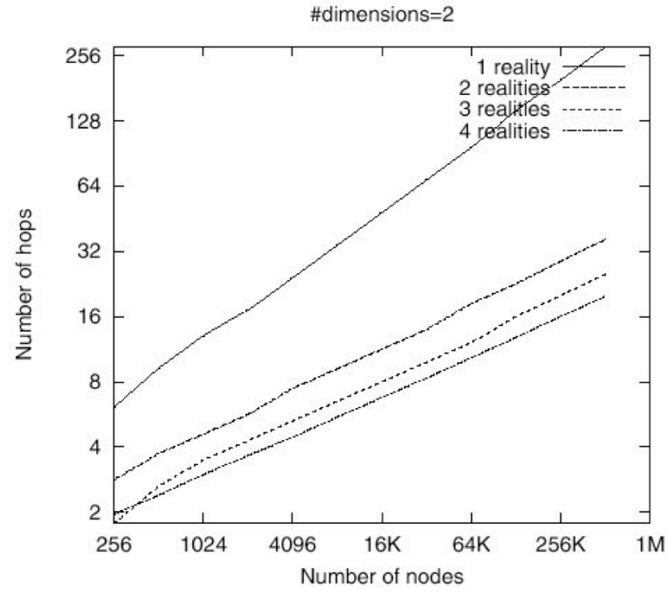
- r different coordinate spaces
  - Peers hold a zone in each of them
  - Creates r replicas of the (key, value) pairs and increases robustness
  - Reduces path length as search can be continued in the reality where the target is closest



CAN has two ways of how failure resilience can be increased by creating replicas. One possibility is to manage r different coordinate spaces at the same time, such that each node has a zone in each of them. This approach is illustrated here. It not only increases resilience, as for every data item r replicas are created, but at the same time reduces search cost as there is a higher probability that the search starts already close to the target with an increasing number of realities.

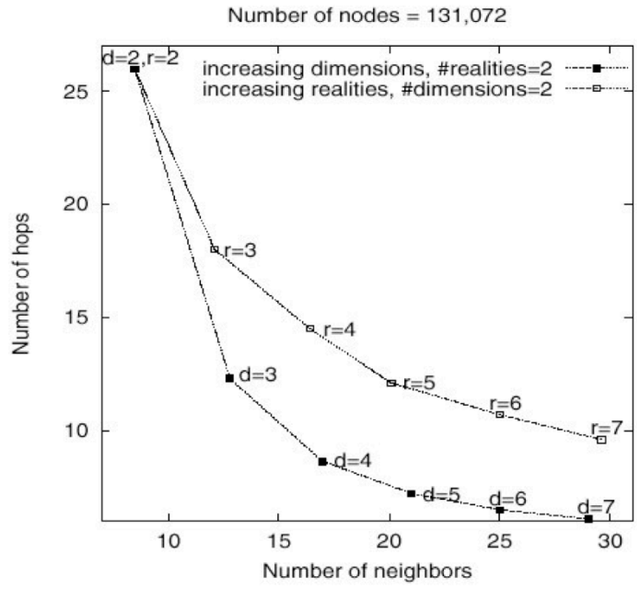
The other possibility to increase resilience, similar to the use of replication in other structured overly networks, is to assign multiple peers to the same zone and to split the zone only if a maximum occupancy is reached. All nodes know each other within the zone, but only one of the neighbors in the neighboring zone.

CAN Path Length



Experimental results show that even with low dimensionality the path length of a search (here #hops) is fairly short. It further improves when multiple realities are used. Note that the axis are of logarithmic scale.

Increasing Dimensions and Realities



Increasing the dimension or the number of realities reduces dramatically the length of the search paths, while it increases the number of neighbors that need to be stored (and changed in case of updates).

Questions

- When adding n peers to CAN the number of zones
  1. Is exactly n
  2. It depends what the keys of the peers were
  3. It depends on the dimensionality of the key space

- In CAN, for a fixed dimensionality d>2, when moving from 1 to 2 realities
  1. The number of entries in the routing table increases by 2
  2. The number of entries in the routing table increases by d
  3. The number of entries in the routing table doubles

Example 4: Dynamical Clustering (Freenet)

- FreeNet Background
  - P2P system which supports publication, replication, and retrieval of data
  - Protects anonymity of authors and readers: infeasible to determine the origin or destination of data
  - Nodes are not aware of what they store (keys and files are sent and stored encrypted)
  - **Uses an adaptive routing and caching strategy**
- Index information maintained at each peer

Key	Data	Address
8e47683isdd0932uje89	ZT38hwe01h02hdhgdzu	tcp/125.45.12.56:6474
456r5wero04d903iksd0	Rhweui12340jhd091230	tcp/67.12.4.65:4711
f3682jkjdn9ndaqmmxia	eqwe1089341ih0zuhge3	tcp/127.156.78.20:8811
wen09hjfdh03uhn4218	erwq038382hjh3728ee7	tcp/78.6.6.7:2544
712345jb89b8nbopledh		tcp/40.56.123.234:1111
d0ui43203803ujoejqhh		tcp/128.121.89.12:9991

FreeNet is a P2P system that has been developed in parallel to Gnutella around 2000 and has become popular, in particular for sharing content while protecting anonymity. It's design goals have been

1. Providing an efficient search mechanism, that limits the number of messages generated by a search.
2. A replication mechanism to disseminate data over the network and thus increase performance and robustness.
3. Protection of anonymity, i.e. peers are not aware who provides which data. Peers even don't know what they store themselves as the data is encrypted. This can be an advantage, in particular with respect to liability, but it could also be understood as a disadvantage as it leads to a lack of "social" control.

We will not further discuss the anonymity aspect of FreeNet, but rather its interesting approach to search. It can be considered as hybrid between the approach taken by unstructured overlay networks and structured overlay networks. Therefore it is also called sometimes a loosely structured overlay network.

As in other structured overlay networks in FreeNet each node stores some index information. In addition it stores also some data related to the index information, in order to provide this information directly and not by referring to another peer. All this information is held in a table as shown above. The size of the table (or cache as it is called in FreeNet for reasons we will see next) is limited and usually many more (key,address) pairs can be cached than (key,data,address) triples.

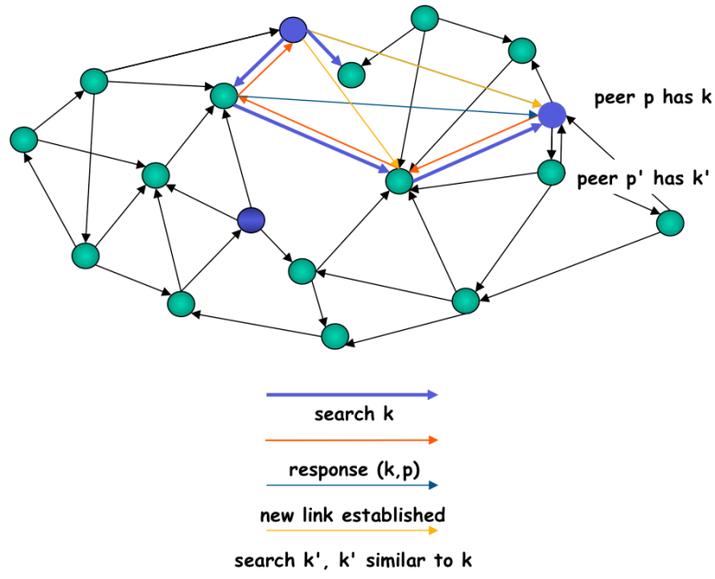
FreeNet Routing

- If a search request arrives
  - Either the data is in the table
  - Or the request is forwarded to the addresses with the most similar keys (lexicographic similarity, edit distance) till an answer is found or TTL reached (e.g. TTL = 500)
- If an answer arrives
  - The key, address and data of the answer are inserted into the table
  - The least recently used key and data is evicted
- Quality of routing should improve over time
  - Node is listed under certain key in routing tables
  - Therefore gets more requests for similar keys
  - Therefore tends to store more entries with similar keys (clustering) when receiving results and caching them
  - Dynamic replication of data

The FreeNet search mechanism contains elements that we know from unstructured overlay networks: if a search request arrives it checks first it's cache whether it contains the response. It also caches responses that it sees from other peers. This is similar to caching in unstructured overlay networks. When the answer is not found locally, the request is forwarded to one other peer. This is comparable to the random walker model in unstructured overlay networks. However, there is one important difference: FreeNet does not chose a random neighbor for forwarding the request, but the one that holds the lexicographically closest key to the search key in the cache. Why is this useful ?

When an answer arrives FreeNet caches it using an LRU strategy. Thus peers tend to cluster together data with similar keys and heuristically therefore future request should be routed more quickly toward those keys. Both effects, the clustering of keys and the dynamic replication of data eventually improve the search latency.

FreeNet Routing



This example illustrates of how the FreeNet network structure changes with each query that is processed. Since the responses are cached along each node, when the answer is returned, new links between the node holding the result and the nodes along the request paths are established. This creates new, shorter paths, for future requests, which are similar (or the same).

Freenet: Inserting Content

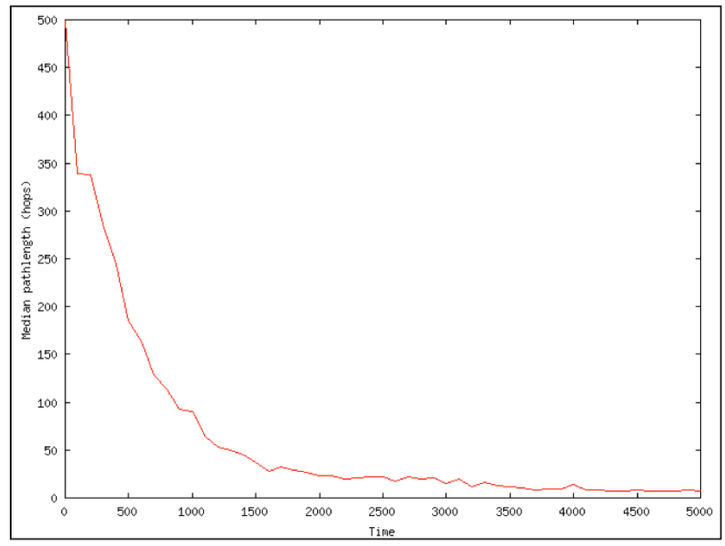
- First a the key of the content (file) is calculated
- An insert message with this proposed key and a hops-to-live value is sent to the neighbor with the most similar key
- Then every peer checks whether the proposed key is already present in its local store
  - yes ⇒ return stored file (original requester must propose new key)
  - no ⇒ route to next peer for further checking (routing uses the same key similarity measure as searching)
  - continue until hops-to-live are 0 or failure
- Hops-to-live is 0 and no collision was detected ⇒ insert file along the path established by insert message

Insertion of new content proceeds similarly as search. Only care has to be taken that the key of the new content does not already exist. Therefore the proposed key is first searched for and only if no collision is detected, the content is inserted under this key. The insertion is again performed along the search path that has been traversed while checking the key. This ensures that the data item is immediately replicated in the network and thus highly available.

Freenet: Evolution of Path Length

- 1000 identical nodes
- max 50 data items/node
- max 200 references/node
- Initial references: (i-1, i-2, i+1, i+2) mod n

- each time-step:
  - randomly insert
  - TTL=20
- every 100 time-steps: 300 requests (TTL=500) from random nodes and measure actual path length (failure=500).



median path length 500 → 6

The effects of caching and clustering of keys in FreeNet can be shown by simulation experiments. This is a typical simulation result obtained with 1000 peers. Every peer stores at most 50 data items and 200 references (index items). Over time (each time step represents one query or one insert), the median query path length dropped from an initial 500 hops to 6 hops. Initially the nodes store their neighbors with at most distance two on a ring (see the graph for illustration). At each time step random inserts are performed with time-to-live 20. After each 100 steps it is tested how good the quality of the access structure is: 300 requests with maximal time-to-live of 500 are processed. The graph shows of how long the search path (maximal 500) is and one can see that it rapidly decreases to a low number (6), which suggests that the access structure is indeed scalable after a bootstrap phase.

Questions

- In FreeNet the routing table is updated
  1. When a search request message arrives
  2. When a query answer message arrives
  3. When an insert file message arrives

- For which of the following structured overlay networks the length of a search path is always guaranteed to be shorter than the length of the longest key
  1. P-Grid
  2. CAN
  3. FreeNet

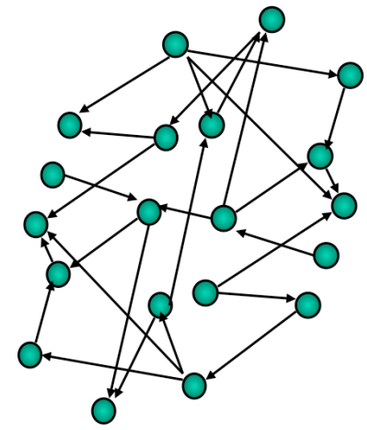
Comparison

	Paradigm	Search Type	Search Cost (messages)
Gnutella	Breadth-first search on graph	String comparison	$2 * \sum_{i=0}^{TTL} C * (C-1)^i$
Freenet	Depth-first search on graph	Equality	O(Log n) ?
Chord	Implicit binary search trees	Equality	O(Log n)
CAN	d-dimensional space	Equality	O(d n^(1/d))
P-Grid	Binary prefix tries	Prefix	O(Log n)

This tables summarizes the key properties with respect to search cost of the different structured overlay network approaches and compares them to unstructured overlay networks.

5. Small World Graphs

- Each P2P system can be interpreted as a directed graph (overlay network)
  - peers correspond to nodes
  - routing table entries as directed links
- Task
  - Find a decentralized algorithm to route a message from any node A to any other node B with few hops compared to the size of the graph
  - Requires the **existence of short paths** in the graph



Over time many different designs for structured overlay networks have been developed. Many of them exhibit lots of similarities, so that an interesting question is whether there exist some underlying principles that could explain these similarities. Interestingly such principles are found in the theory of small world graphs which initially developed in social sciences, which at a second glance is less surprising as peer-to-peer systems are systems that a have strong social dimensions.

From an abstract perspective we can consider each overlay network as a graph structure and search in overlay networks as a routing algorithm in the graph.

Milgram's Experiment

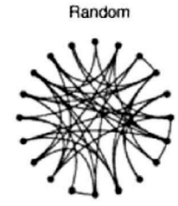
- Finding short chains of acquaintances linking pairs of people in USA who didn't know each other;
  - Source person in Nebraska
  - Sends message with first name and location
  - Target person in Massachusetts.
- Average length of the chains that were completed was between 5 and 6 steps
- "Six degrees of separation" principle

- BIG QUESTION:
  - WHY there should be short chains of acquaintances linking together arbitrary pairs of strangers???

Regarding search in such graphs Stanely Milgram performed in 1967 an interesting experiment with humans as described. The insight of the experiment was that any pair of complete strangers seems to have a very short chain of acquaintances linking them.

Random Graphs

- For many years typical explanation was - random graphs
  - Low diameter: expected distance between two nodes is $\log_k n$
  - k is the average outdegree and n the number of nodes
  - When pairs or vertices are selected uniformly at random they are connected by a short path with high probability
- But there are some inaccuracies
  - If A and B have a common friend C it is more likely that they themselves will be friends! (clustering)
  - Many real world networks (social networks, biological networks in nature, artificial networks – power grid, WWW) exhibit this clustering property
  - Random networks are NOT clustered.



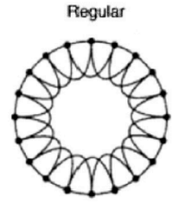
This property was initially explained by the theory of random graphs. It tells us that in a random graph any pair of nodes has a short connection of logarithmic length in the number of nodes of the graph. However, since acquaintances are not purely random, but rather clustered, this explanation is not very convincing. This initiated research by Watts and Strogatz published in 1998, that tried to come up with a better explanation.

Clustering

- Clustering measures the fraction of neighbors of a node that are connected themselves
- Regular Graphs have a high clustering coefficient
  - but also a high diameter
- Random Graphs have a low clustering coefficient
  - but a low diameter
- Both models do match some properties expected from real networks!

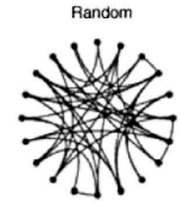
Regular Graph (k=4)
Long paths
- $L \sim n/(2k)$

Highly clustered
- $C \sim 3/4$



Random Graph (k=4)
Short path length
- $L \sim \log_k N$

Almost no clustering
- $C \sim k/n$



Definitions:

The *distance* between to nodes in a undirected graph is the length of the shortest path connecting them.

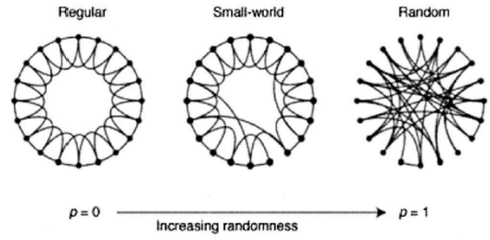
The *diameter* of the graph is the maximum distance among any pair of nodes in the graph.

The *local clustering coefficient* of a node in an undirected graph corresponds to the fraction of neighbors that are connected. More precisely when N is the set of neighbors of a node i of size |N|=k, and E is the number of edges among those neighbors, then the local clustering coefficient is C_i = 2 |E| / k (k-1), since there exist k (k-1) / 2 potential edges among neighbors.

The *global clustering coefficient* is then the average of all local clustering coefficients over all nodes, i.e. C = 1/n sum_i^n C_i, for a graph with n nodes.

Small-World Networks

- Random rewiring of regular graph (by Watts and Strogatz)
  - With probability *p* rewire each link in a regular graph to a randomly selected node
  - Resulting graph has properties, both of regular and random graphs
    - High clustering and short path length
  - FreeNet has been shown to result in small world graphs



Watts and Strogatz found that by introducing few random links in a highly regular graph with high diameter, is sufficient to arrive at graphs with low diameter. At the time this fact was not known, and introduced a whole new class of graphs, which were called small world graphs.

Question

- The local clustering coefficient is the probability that two of my friends are also friends. If I have 10 friends and among them 15 friendships exist, my local clustering coefficient is
  1. 1/6
  2. 1/3
  3. 2/3
  4. 3/2

- A random graph has
  1. High clustering and low diameter
  2. High clustering and high diameter
  3. Low clustering and low diameter
  4. Low clustering and high diameter

Search in Small World Graphs

- BUT! Watts-Strogatz can provide a model for the structure of the graph
  - existence of short paths
  - high clustering
- It does not explain **how** the shortest paths are found
  - also Gnutella networks are small-world graphs
  - why can search be efficient, e.g. in FreeNet?

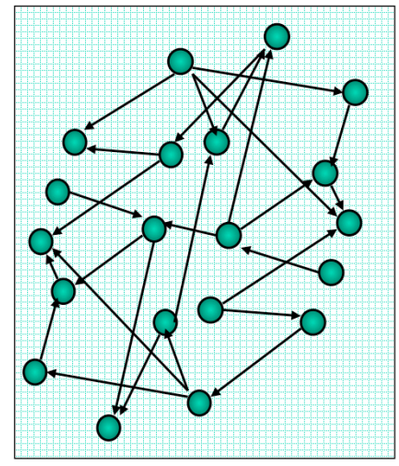
The issue that is not explained by the theory of Watts and Strogatz is how the shortest paths can be found!

P2P Overlay Networks as Graphs

- Each structured P2P overlay network can be interpreted as a directed graph …
  - peers correspond to nodes
  - routing table entries as directed links

… embedded in some space
  - P-Grid: interval [0,1]
  - Chord: ring [0,1)
  - CAN: d-dimensional torus
  - FreeNet: strings + lexicographical distance

- Task
  - Use **greedy routing** to route a message from any node A to any other node B with few hops compared to the size of the graph
  - Greedy routing: at each step try to minimize distance to the target



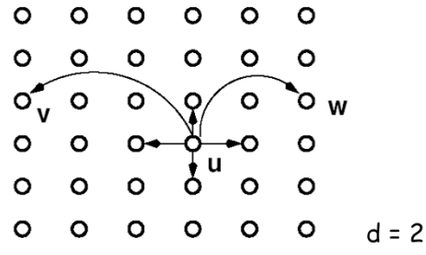
In order to enable the search of the shortest path in a small world graph we need a second element: the notion of distance. Indeed considering all designs of structured overlay networks we see that they all use some (metric) space in to order to route messages to the target in a greedy fashion.

Kleinberg's Small-World Model

- Kleinberg's Small-World's model
  - Embed a graph into an d-dimensional grid
  - constant number p of short range links (neighborhood)
  - q long range links: choose long-range links from u to v such that

$$P[link(u,v)] \propto dist(u,v)^{-r}$$

- Importance of r !
  - Decentralized (greedy) routing performs best iff. r = d = dimension of space



This idea has been formalized by another well known work, the small world model developed by Jon Kleinberg. It considers graphs that are embedded into regular grids of a given dimensionality d. In the model he shows that even if nodes have only a single randomly chosen long range link in addition to the links to their immediate neighbors in the grid (similar as in CAN) efficient greedy routing can be performed in the resulting graph. He introduced a probabilistic model to choose these long range link, depending on the distance among nodes and a design parameter r.

He showed what is the optimal way to choose these long range links: the probability of choosing a node should be proportional to 1/dist^r with r=d, where dist is the distance to the node and d the dimensionality of the space.

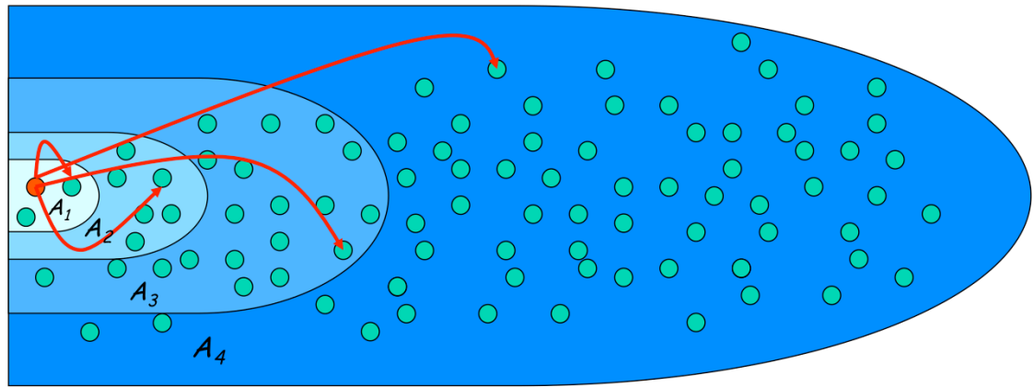
Influence of r

- Optimal value: r = d = dimension of the space
- If r < d: we tend to choose more far away neighbors
  - greedy routing can quickly approach the neighborhood of target, but then slows down till finally reaches target itself
- If r > dim: we tend to choose more close neighbors
  - greedy routing finds quickly the target in it's neighborhood, but reaches it slowly if it is far away
- If r = 0: long range contacts are chosen uniformly
  - random graph theory proves that there exist short paths between every pair of vertices, BUT there is no decentralized algorithm capable finding these paths efficiently
- Routing cost
  - With 1 long range link: O(log^2 n)
  - With log n long range links: O(log n)

Only if precisely r=d, the resulting network will perform optimally.

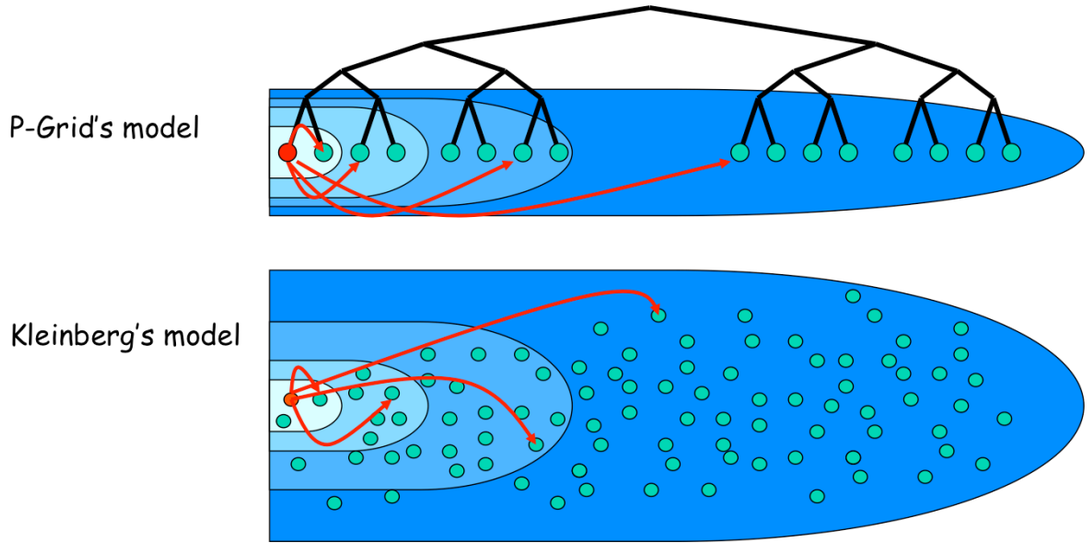
Structured Overlay Networks and Kleinberg model

- Given node u
  - we can partition the remaining peers into sets $A_1, A_2, A_3, \ldots, A_{logn}$
  - A_i consists of all nodes whose distance from u is between 2^i and 2^{i+1}
  - each long range contact of u is equally likely to belong to any of the sets A_i *(r=d)*
  - if $q = log\ n$: on average each node will have a link to each set A_i



To better understand the effect of choosing long range neighbors as introduced by Kleinberg, consider the illustration given above. It indeed says that among partitions of the space, at exponentially increasing distances links are chosen uniformly. This reminds of the approach of how long rang links are chosen in structured overlay networks such as P-Grid or Chord!

Structured Overlay Networks and Kleinberg model



This figure shows how the Kleinberg model corresponds to the structure of the P-Grid network.

Indeed it seems that many of the structured overlay network designs can be nicely mapped to the Kleinberg model and that there exists a common explanation for the typical performance.

Conclusions

- Kleinberg's model
  - there is no decentralized algorithm capable performing effective search in the class of SW networks constructed according to Watts and Strogatz
  - J. Kleinberg presented the infinite family of Small World networks that generalizes the Watts and Strogatz model and shows that decentralized search algorithms can find short paths with high probability
  - there exist only one unique model within that family for which decentralized algorithms are effective.

- With respect to overlay networks
  - Many of the structured P2P overlay networks are similar to Kleinberg's model (e.g. Chord, randomized version, q=log N, r=1)
  - Unstructured overlay networks also fit into the model (e.g. Gnutella q=5, r=0)
  - Some variants of structured P2P overlay networks are having no neighborhood lattice (e.g. P-Grid, p=0)
  - Extensions to spaces beyond regular grids are possible (e.g. arbitrary metric spaces)

Here we summarize the findings on small world graphs and how the theory relates to the different overlay networks we have discussed, including unstructured ones.

Question

- In a three-dimensional Kleinberg small world network with log n long range links the search cost is
  1. Log n
  2. Log^2 n
  3. Log^3 n

6. P2P Data management

- Frequently the resources managed in P2P systems is data
  - Metadata on resources (files, services)
  - P2P data management
- Structured Overlay Networks particularly suitable
  - Distributed Index
- Managing data is different from resource identifier lookup
  - (random) identifiers uniformly distributed
  - Only equality searches
- Problems in P2P data management
  - Range queries
  - Multiple attribute queries
  - Join queries

Initially most peer-to-peer systems were developed with the design goal of supporting exact key lookups of resources. This is in particular true for structured overlay networks that map both application keys and peer identifiers into a common identifier space. These systems perform, using terminology introduced earlier, a hash partitioning of the data, which consists of a single table with the structure (key, value). The value is a reference to access the resource or sometimes the resource itself, and the keys are identifiers of resources.

Often the keys correspond to some application data and carry some meaning, that goes beyond the pure identification of a resource. For example, in the most common case where a filename is used as key, the key might contain substrings useful for search. This is even more true when structured overlay networks are explicitly used to index structured data. Then even simple searches might require search predicates that require more than equality searches. Examples of such searches are range queries (for example for numerical attributes), queries involving multiple attributes, and join queries.

Example

- Searching metadata on content

fileid	title	author	date	sold
vfa98qm	Beat It	Michael Jackson	1982	1.25
x9dg8fa	Like Toy Soldiers	Eminem	2005	1.05
h7ab43a	You're Beautiful	James Blunt	2007	1.24
9fda0fm	A Hard Day's Night	Beatles	1964	4.89

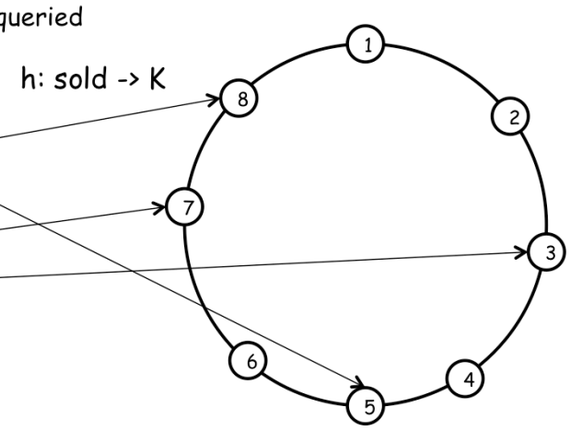
- Range query: searching for size (e.g. 1.0 < sold < 1.5)
- Multiple attribute search: e.g. author ="Michael Jackson" and date = 1982
- Joins (e.g. T1.date = T2.date-1 and T1.sold > 1.0 and T2.sold > 1.0)

This simple example illustrates different kinds of searches that should be typically supported, when managing metadata on contents that are shared in a peer-to-peer system.

1. Range Query Processing

- Most structured overlays generate identifiers using a uniform randomized hash function h
  - Hash partitioning
  - Data is approximately uniformly distributed in the identifier space
  - Order among indexed values is not preserved
  - Ranges cannot be efficiently queried

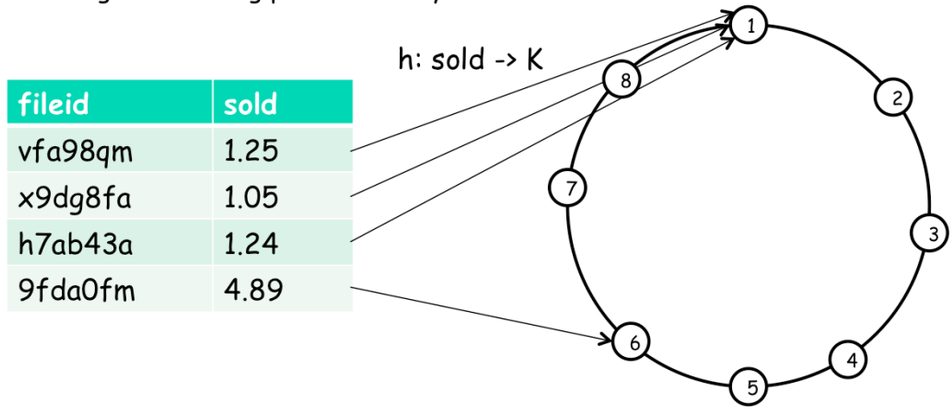
fileid	sold
vfa98qm	1.25
x9dg8fa	1.05
h7ab43a	1.24
9fda0fm	4.89



A structured overlay network can be used to index an attribute, as shown in the example for the attribute “sold”. When mapping values to the key space (e.g. the unit circle as illustrated for the example for a Chord network with 8 peers), usually a randomized hash function is used. As a result the order relationship among the original values is lost, and one important type of queries, range queries, can no longer be supported efficiently.

Order-Preserving Hashing

- If for values v1 < v2 the hash function h generates keys h(v1) < h(v2), then it is **order-preserving**
  - Range partitioning
  - Ranges can be easily queried
  - Storage load among peers unevenly distributed -> bottlenecks



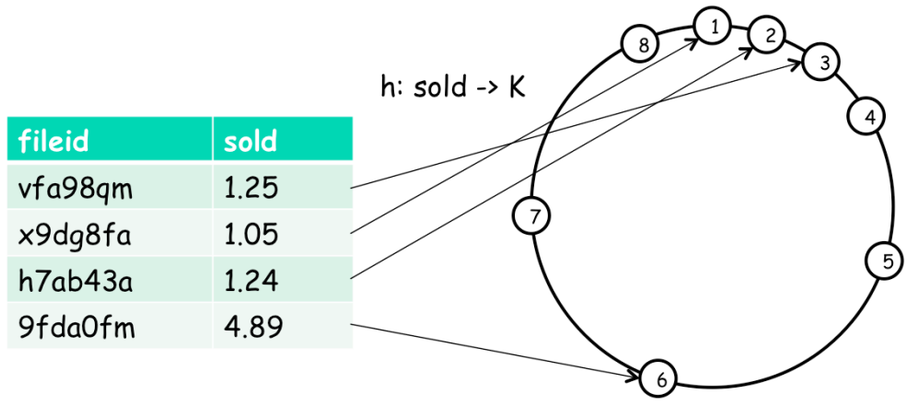
fileid	sold
vfa98qm	1.25
x9dg8fa	1.05
h7ab43a	1.24
9fda0fm	4.89

In order to address this problem one may use a order-preserving hash function. In the earlier terminology we used for distributing relational tables this results in a range partitioning of the table over the peers. Now answering range queries becomes straightforward: one searches the lower (upper) bound of the query range and then traverses the peers using successor (predecessor) links till the upper (lower) bound is reached.

Assuming that the peers are approximately uniformly distributed over the key space it may happen that the data distribution over the peers might be highly non-uniform. For many types of data it is known that their values are highly non-uniformly distributed, e.g. they might follow a Zipf or powerlaw distribution. As a result a few peers would carry most of the workload in the system, e.g. for providing storage space, and for using network and processing resources for answering requests.

Approach 1: Load Balancing

- Solution: peers change their position in the key space so that each partition carries approximately the same load



fileid	sold
vfa98qm	1.25
x9dg8fa	1.05
h7ab43a	1.24
9fda0fm	4.89

A possible solution might be to distribute also the peers non-uniformly over the key space, i.e. to populate those regions of the key space more densely with peers where more data will be mapped. This results in a better balancing of the load of the peers. The problem here is, that since there is no (global) knowledge available on how the application data is distributed, and furthermore the peers are not coordinated centrally, it seems for the peers difficult to figure out of how to achieve such a distribution.

Load Balancing Algorithms

- Principle
  - Peers contact other peers and if they detect big imbalance the peer with less load changes position and shares the load with the overloaded peer
  - Example algorithm (random load balance)

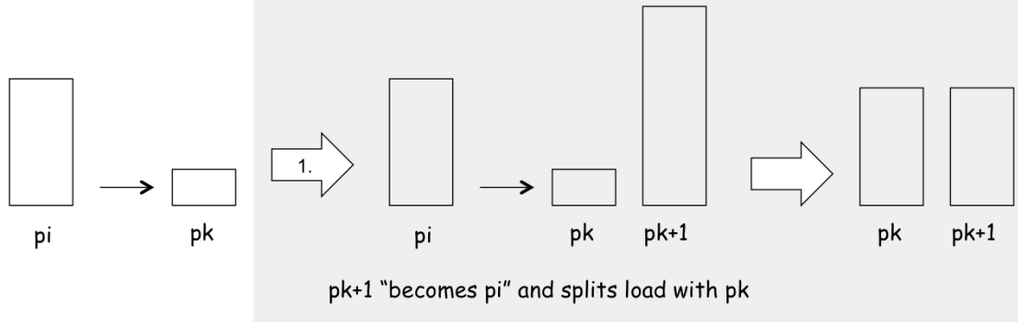
    A peer contacts a random peer
    If delta load(p_i) > load(p_k) then start load balance
    If load(p_{k+1}) > load(p_k) better to load balance with neighbor of p_k
        p_i becomes p_{k+1}
    If p_i and p_k are neighbors balance out the load
    Otherwise
        p_k transfers its data to p_{k+1} (which is not highly loaded)
        p_i and p_k balance out the load

  - Proven to limit load imbalances in the network

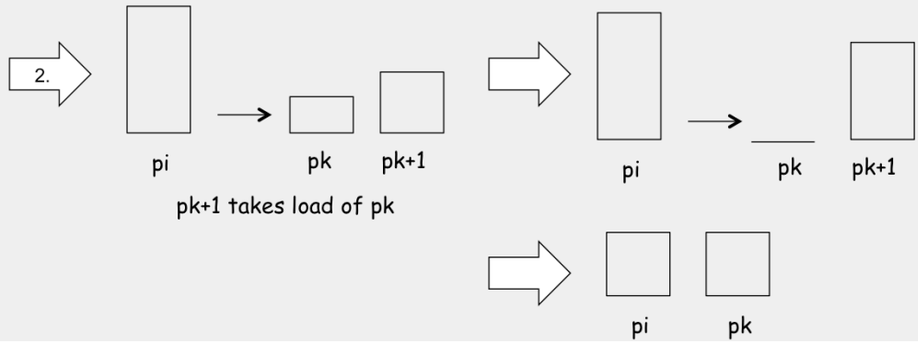
In order to address the problem of load balancing in a peer-to-peer system different decentralized algorithms have been proposed. They follow all similar principles: peers observe their own load and compare it with the load of a few other peers, which might be either direct neighbors or randomly selected peers, that are found for example using a random walk. If the load among two peers becomes too different then the peers decide to balance locally their load by changing their position in the key space. Here we show one such algorithm as an example. It works as follows:

The algorithm uses periodic sampling. Each peer performs occasionally Algorithm 2 at random using a load imbalance threshold 0 < delta< 1/4. In this algorithm when the peers contacts another random peer and the load imbalance ratio is beyond the threshold (line 1) it first checks if the next higher neighbor of the contacted peer has an even higher load than the peer itself. In that case or if the both peers already have been neighbors they balance the load among each other (lines 8-9). Otherwise the contacted peer pk sheds its load to its neighbor, changes it position as new neighbor of pi and balances its load with pi (lines 11-13).

Illustration



pk+1 "becomes pi" and splits load with pk

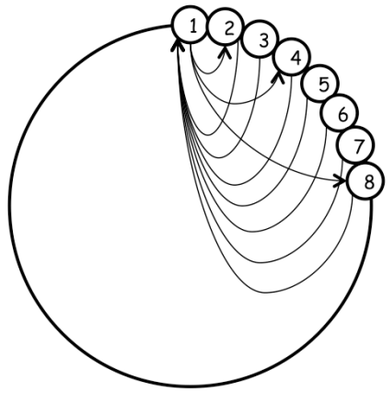


pk+1 takes load of pk

pi splits load with pk

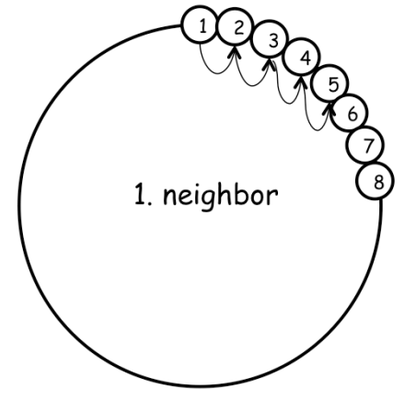
Approach 2: Constructing Overlays for Non-Uniform Distribution

Using the standard Chord approach:
routing bottlenneck (peer 1)



Using hop count:

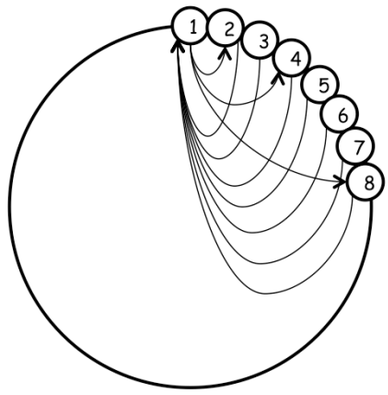
$i+1^{th}$ neighbor is the i^{th} neighbor of the i^{th} neighbor



After having achieved load balance peers, however, face a second problem. With an unbalanced distribution of peers in the identifier space the standard algorithm to construct the routing tables results in highly imbalanced distribution of links. In the example shown above all links of peers 2-8 point to peer 1. As a result peer 1 becomes overloaded in routing and routing cost increases. In order to fix this problem one has to distinguish between the distance among two peers in the key space (which is used normally in order to construct the routing tables) and the distance among two peers in terms of the number of hops between them, when traversing the peer network using successor links only. For the construction of load-balanced routing tables this “hop distance” is essential and therefore to be used. This can be performed in an incremental way as follows. The first long-range neighbor of a peer is simply its successor peer. To determines its i + 1th neighbor, a peer contacts the ith neighbor in its routing table and requests it's ith neighbor p. Then it sets it‘s own i + 1th neighbor to this peer p. This procedure is repeated till the long-range link points beyond the peer itself. The process of construction is illustrated on the following slides.

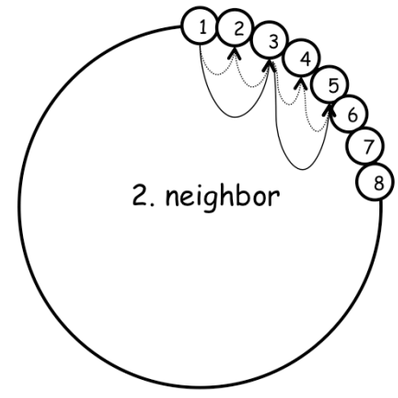
Constructing Overlays for Non-Uniform Distribution

Using the standard Chord approach



Using hop count:

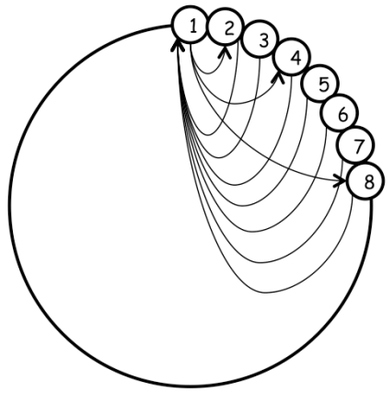
$i+1^{th}$ neighbor is the i^{th} neighbor of the i^{th} neighbor



Here the second generation of long-range links is created.

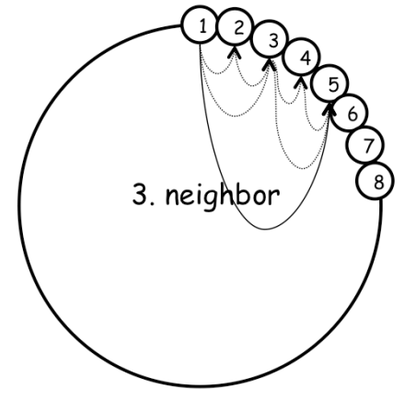
Constructing Overlays for Non-Uniform Distribution

Using the standard Chord approach



Using hop count:

$i+1^{th}$ neighbor is the i^{th} neighbor of the i^{th} neighbor



Here the third (and final) generation of long-range links is created.

Questions

- Order-preserving hashing is used
  1. To load balance the keys in a peer-to-peer overlay network
  2. To support efficient processing of range queries
  3. To avoid routing bottlenecks with non-uniformly distributed peers on a Chord ring

- When constructing a Chord ring using the hop count distance measure
  1. Load balancing should be used to balance the resulting different storage loads among peers
  2. The routing load among peers will be necessarily uniformly distributed for a random query workload
  3. Range queries can be efficiently processed, even when using a random hash function

2. Managing Multiple Attributes

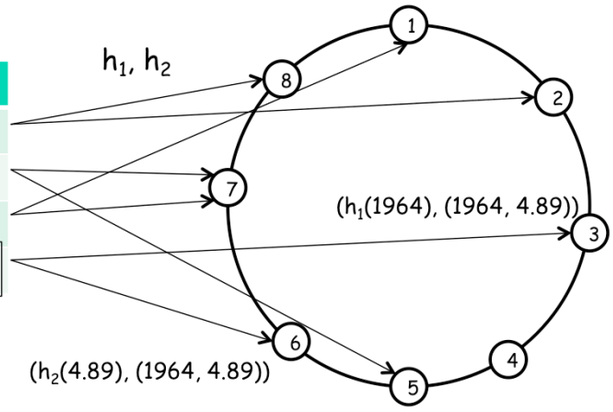
- Assume we want to search over multiple attributes a_i, i=1,…,d
  - Conjunction of conditions on some of the attributes
- Simple Solution
  - Map each attribute value v_i using a order-preserving hash function h_i
  - Store the tuple (v_i, (v_1,…,v_d)) at the peer that is responsible for $h_i(v_i)$
- Evaluating a condition on multiple attributes
  - Choose first the most selective attribute to search the peer (or peers) that store the value
  - Retrieve all the tuples stored
  - Filter them according to the other conditions

Applications for resource management frequently identify resources by multiple attributes, as shown in the earlier example. We study now the problem of supporting search including range queries over multiple attributes. We assume that a fixed number of attributes a1, …, ad is given. We also assume that the number of attributes d is relatively small. Therefore we are searching in a low-dimensional space. Generalizing the one-dimensional case of range queries, we assume that the most general query type are multi-dimensional range queries, where a range condition is given for each attribute and the different conditions are connected by a conjunction (conjunctive query).

A simple approach for mapping multiple attributes to an overlay network is to store a multi-dimensional value (v1,…,vd) as follows in the peer-to-peer overlay network. Each of the values vi is normalized to the same range and then mapped by an order-preserving hash function to a peer in a one-dimensional Chord ring. The peers maintain for each attribute ai a separate list in which they store the pairs (vi,(v1,…,vd)) for the values vi they are responsible for. A naive approach to answer multi-dimensional range queries with this storage scheme, is to query for each attribute separately and then compute the intersection of the received results at the query originator. This is however in general very costly and not necessary, since the values of the other attributes are known when looking up the value of a specific attribute ai. Therefore a more efficient approach is to first select the attribute for which the query condition is the most selective, and then iteratively traverse the query range, while filtering out those tuples that do not satisfy the search condition.

Example

	h_1	h_2
fileid	date	sold
vfa98qm	1982	1.25
x9dg8fa	2005	1.05
h7ab43a	2007	1.24
9fda0fm	1964	4.89

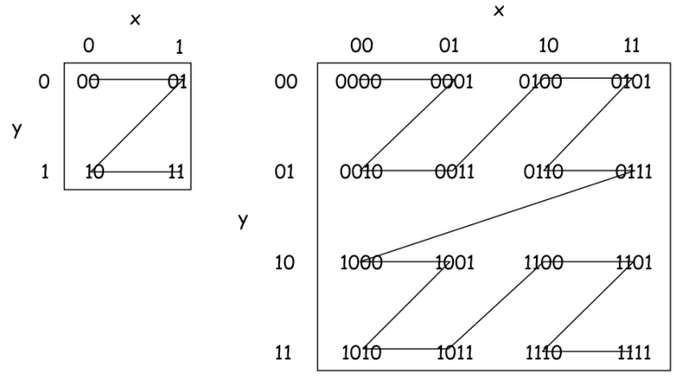


- Drawback
  - All attribute values are replicated d times
  - Can we do better?

This example illustrates of how the same data is replicated on different peers. When indexing multiple attributes in a structured overlay network.

Mapping to One-dimensional Space

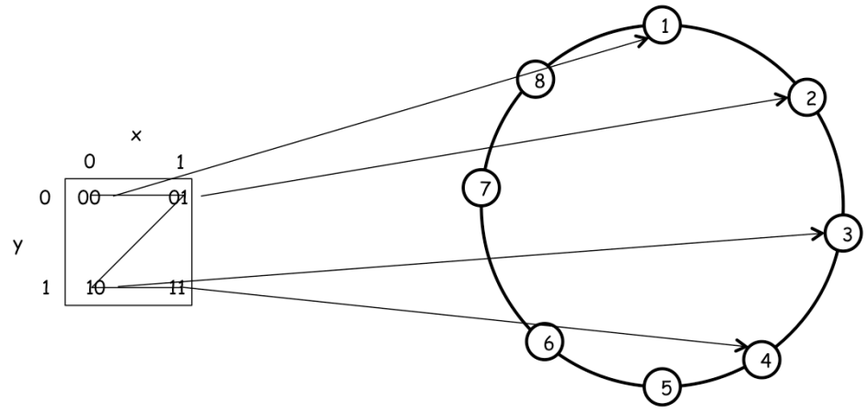
- Use a space filling curve to map from d-dimensional attribute space to one-dimensional space
- Example: Mapping from two to one dimension, z-curve
  - Generated by shuffling the bits of the x and y values



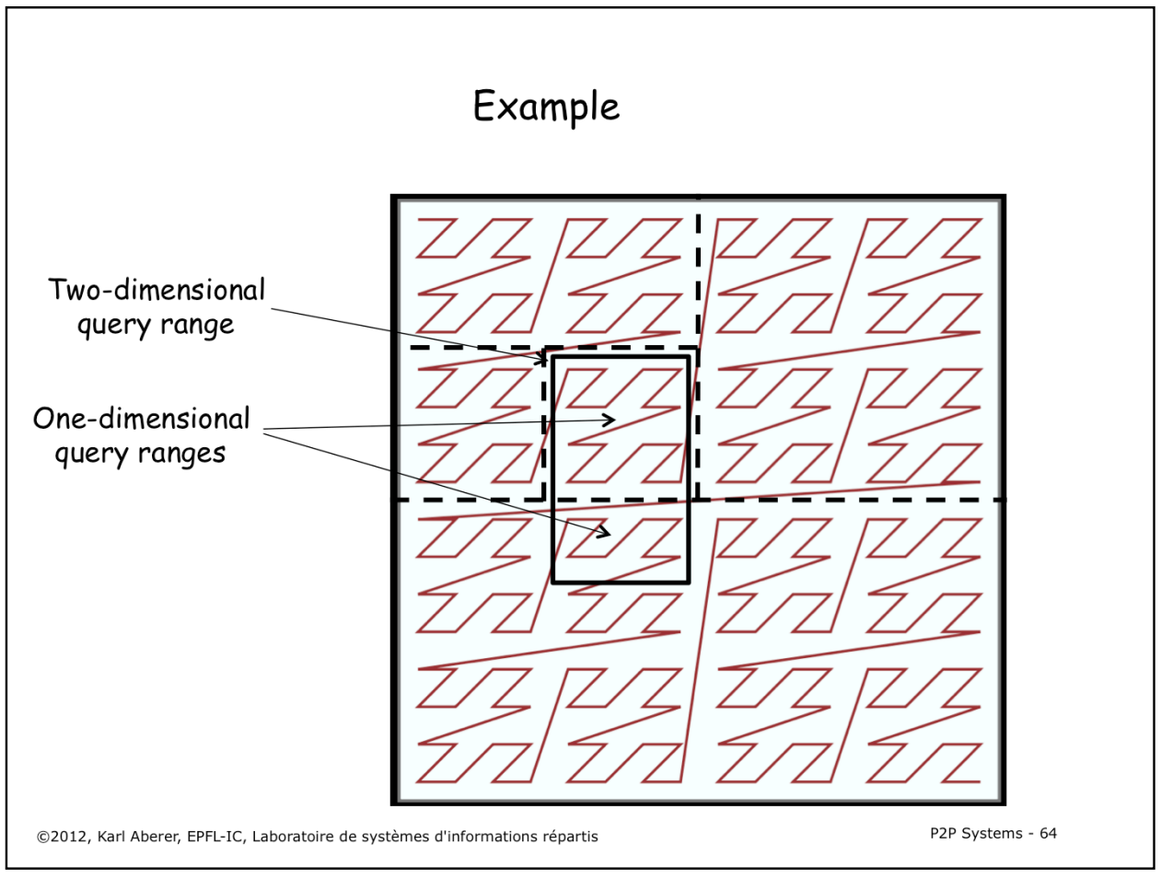
An alternative approach to index multiple attribute in an overlay network, is to map the d-dimensional attribute space to a one-dimensional space using a space-filling curve. One example of such a mapping by a space-filling curve is the z-ordering. The mapping is obtained by shuffling the bits of a binary representation of the attribute values. More precisely, if attribute ai has a binary representation bi1 … bik, then a tuple (a1,…,ad) is mapped to the one-dimensional value b11 b21 … bd1 b12 … bd2 … b1k … bdk. This mapping preserves locality as tuples with similar values along all the dimensions will be mapped to close values in the one-dimensional space. Once the data has been mapped to the one-dimensional space it is distributed over the nodes the data can be distributed of a one-dimensional structured overlay network.

Indexing and Query Processing

- Range query processing
  - A multi-dimensional range query is transformed into a set of one-dimensional queries
  - recursively bi-partitions the multi-dimensional space till
    - either the subspace is completely contained in the query range
    - or it has an empty intersection with the query range



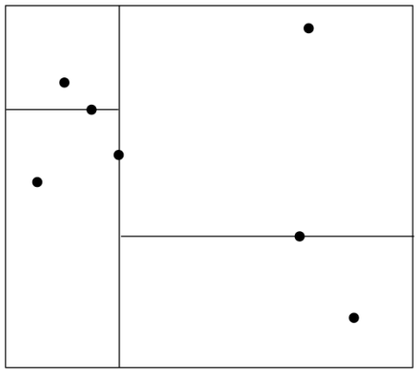
Processing range queries over multiple attributes is then performed in two steps. First the multi-dimensional query is transformed into a set of one-dimensional queries that is guaranteed to contain all query answers. The transformation algorithm recursively bi-partitions the multi-dimensional space till either the subspace is completely contained in the query range or it has an empty intersection. In a second phase each one-dimensional range query is evaluated separately by traversing the peers that fall into the query range.

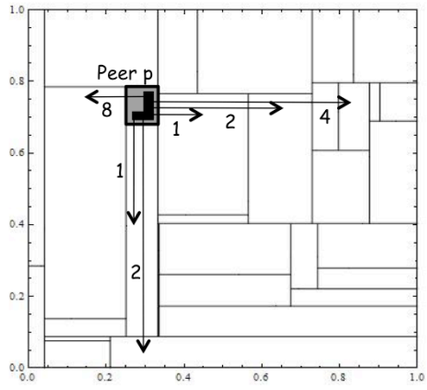


This example shows of how a two-dimensional query range is split using bipartitioning of the two-dimensional space into two one-dimensional ranges (in the z-ordering).

Using Space Partitioning

- Use multi-dimensional structured overlay networks (like CAN)
  - Identifier space has same dimensionality as attribute space
  - Recursively perform balanced splits (kd-Tree)
  - Construct routing table using hop count





A third method to support multi-attribute queries in P2P networks is based on the observation that some structured peer-to-peer systems (like CAN) use a multi-dimensional key space. Thus the attributes values can (after normalization, e.g. to the interval [0,1]) be directly mapped into the key space of the overlay network and thus stored by the peer that is responsible for managing the subspace the mapped value falls into. Using this approach the following method has been devised for managing multiple attributes. It is based on an adaptation of a multi-dimensional index structure that is well known from databases, the kd-Tree. The construction of a kd-tee is based on a recursive partitioning of the multidimensional space. For a split the data point is chosen that is the median along the split dimensions. The split dimensions are chosen cyclically. The peers maintain as in CAN direct links with their neighbor regions along all dimensions. For improving routing efficiency (and as extension to CAN) one can add long-range links along each dimension following the same principle as described in the procedure for constructing long-range links for a one-dimensional peer-to-peer system with non-uniform peer distribution. The first long-range neighbor of a peer in dimension i is simply its neigh- boring peer along that dimension. To determines its i + 1th neighbor, a peer contacts the ith neighbor in its routing table of that dimension and requests it's ith neighbor p. Then it sets it's own i + 1th neighbor to this peer p.

Question

- Which of the following techniques for handling multi-attribute queries in overlay networks is (significantly) sacrificing query performance?
  1. Indexing each attribute separately
  2. Encoding the multiple attributes into a single key using a z-curve
  3. Using non-uniform space partitioning

Summary

- Complex query processing in peer-to-peer networks
  - Structured overlay networks are a natural substrate for efficient query processing
  - Comparable to indexing structures in conventional databases
- Challenge
  - Dealing with non-uniform load distribution in the absence of central coordination
  - Breaks many assumptions usually made for structured overlay networks
- Applies many well-known techniques from traditional data management
  - Space partitioning for multi-dimensional indexing
  - Mapping from multi- to one-dimensional space

References

Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, March 2001. http://www.oreilly.com/catalog/peertopeer/.

K. Aberer, M. Hauswirth: "*P2P Systems*", Practical Handbook of Internet Computing, CRC press, 2004

K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, R. Schmidt, J. Wu: " *Advanced Peer-to-Peer Networking: The P-Grid System and its Applications* ", PIK - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems, 26(3), 2003.

Beverly Yang, Hector Garcia-Molina: *Improving Search in Peer-to-Peer Networks*. ICDCS 2002: 5-14

Clip2. *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*. June 15, 2001. http://www.clip2.com/GnutellaProtocol04.pdf

M.A. Jovanovic, F.S. Annexstein, and K.A.Berman. *Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella*. University of Cincinnati, Laboratory for Networks and Applied Graph Theory, 2001. http://www.ececs.uc.edu/~mjovanov/Research/paper.ps

Eytan Adar and Bernardo A. Huberman. *Free Riding on Gnutella*. Technical report. Xerox PARC. September 9, 2000. http://www.parc.xerox.com/istl/groups/iea/papers/gnutella/Gnutella.pdf

Kunwadee Sripanidkulchai. *The popularity of Gnutella queries and its implications on scalability*. February 2001. http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

Matei Ripeanu, Adriana Iamnitchi, Ian T. Foster: Mapping the Gnutella Network. IEEE Internet Computing 6(1): 50-57 (2002)

References

Qin Lv, Pei Cao, Edith Cohen, Kai Li, Scott Shenker: Search and replication in unstructured peer-to-peer networks. SIGMETRICS 2002: 258-259

Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. LLNCS 2009. Springer Verlag 2001. http://www.freenetproject.org/index.php?page=icsi-revised

Theodore Hong. *Performance in Decentralized Filesharing Networks*. Presentation given by at the O'Reilly Peer-to-Peer Conference, San Francisco. February 14-16, 2001. http://www.freenetproject.org/p2p-theo.ppt

Jon Kleinberg. *The Small-World Phenomenon: An Algorithmic Perspective*. Technical report 99-1776. Cornell Computer Science, October 1999. http://www.cs.cornell.edu/home/kleinber/swn.pdf

Heylighen F. (1999): *The Science of Self-organization and Adaptivity*, in: The Encyclopedia of Life Support Systems Heylighen F. (1999): Collective Intelligence and its Implementation on the Web: algorithms to develop a collective mental map, Computational and Mathematical Theory of Organizations 5(3), 253-280.

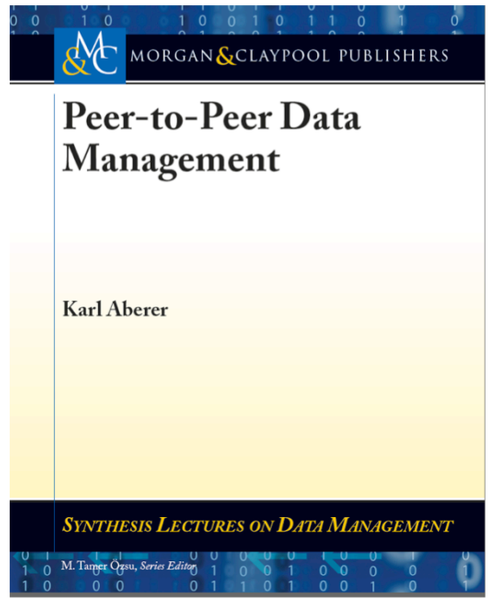
A. Barabasi, R. Albert: *Emergence of Scaling in random Networks*, Science, Vol 286, 1999.

Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan. *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications*. Proceedings of the ACM SIGCOMM, 2001.

Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. *A Scalable Content-Addressable Network*. Proceedings of the ACM SIGCOMM, 2001.

References

- Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan: A multi-attribute addressable network for grid information services. In Journal of Grid Computing, page 184. IEEE Computer Society, 2003.
- David R. Karger and Matthias Ruhl. Simple ecient load-balancing algorithms for peer-to-peer systems. Theory of Computing Systems, 39:787{804, 2006.
- J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In PODS '84: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, pages 181{190, New York, NY, USA, 1984.
- Thorsten Schuett, Florian Schintke, and Alexander Reinefeld. Range queries on structured overlay networks. Computer Communications, 31(2):280 -291, 2008.
- Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In SIGCOMM '04: pages 353{366, New York, NY, USA, 2004.



Download at http://www.morganclaypool.com/doi/abs/10.2200/S00338ED1V01Y201104DTM015
or google for "peer-to-peer data management"