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

Problem 3. Scalability for Large Databases

- Locate data on available storage medium in an efficient manner
- Blocks, file, directory/index
- Provide efficient access to data for specific addressing methods
  (Indexing)
  - Attributes, predicate, query...
  - Data access structure (tree, hash table etc.)
- Example: B+-Tree
  - tree scale match block size of storage systems
  - tree is balanced
  - all operations (search, update) logarithmic.
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
  - tremendous bandwidth consumption for search
  - free riding

- Can we do better?
Efficient Resource Location

- **FULL REPLICATION**
  - High update cost
  - High search cost
  - High maximal bandwidth

- **UNSTRUCTURED P2P OVERLAY NETWORKS** (e.g. Gnutella)
  - Low update cost
  - Low search cost
  - High maximal bandwidth

- **STRUCTURED P2P OVERLAY NETWORKS** (e.g. prefix routing)
  - Low update cost
  - Low search cost
  - Low maximal bandwidth

- **SERVER** (e.g. Napster)
  - High update cost
  - High search cost
  - High maximal bandwidth
Distribution of Index Information

- **Goal**: provide efficient search using few messages without using designated servers
- **Easy**: distribution of index information over all peers, i.e. every peer maintains and provides part of the index information \((k, p)\)
- **Difficult**: distributing the data access structure to support efficient search

Hierarchical P2P networks introduce multiple levels at which nodes take over different roles, in order to support indexing of data. One of the reasons why the index information is concentrated on few nodes, is the possibility to use 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).

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?
To solve the problem of distributing a data access structure recently 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. In the following we will study 4 approaches, each representative for a different paradigm and for a class of related approaches.
A second class of approaches to implementing distributed search trees is based on the idea to distribute a search tree in a scalable manner. This idea underlies a number of different approaches, of which P-Grid is the one we will describe here. In the following we will more precisely consider binary search tries as the tree-based search structure.
In order to make the search tree available in a distributed environment the search tree (the index data) needs to be distributed over the peers (that hold the distributed data). One possibility for doing this is indicated above: peers partition the data space and hold the corresponding data items. In addition they store a part of the search tree that belongs to the path from their leaf to the root. However, partitioning the search tree in the manner shown (which in fact is done for distributed indexing in workstation clusters, for example) 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 Napster uses. 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 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 by itself, 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 each peer, because each 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.
Construction

- **Splitting Approach (P-Grid)**
  - peers meet and decide whether to extend search tree by splitting the data space
  - peers can perform load balancing considering their storage load
  - networks with different origins can merge, like Gnutella, FreeNet (loose coupling)

- **Node Insertion Approach (Chord, CAN, ...)**
  - peers determine their "leaf position" based on their IP address
  - nodes route from a gateway node to their node-id to populate the routing table
  - network has to start from single origin (strong coupling)

- Replication of data items and routing table entries is used to increase failure resilience

With respect to constructing a distributed tree, there exist two different approaches: the standard approach of node insertion allows only to add one node at a time to the network. It works similarly as Chord, by assigning a key to the peer, typically based on its IP number, and then updating the routing table of the new peer and the existing peers correspondingly. This approach bears the same limitations on peer autonomy and required global knowledge as Chord.

An alternative has been proposed in P-Grid: peers locally construct tree structures by performing local splits of the search space. The decision whether and how to split can be governed by additional parameters, such as the current load or the available resources at the peer. In this way networks can start independently and get increasingly merged whenever peers from different networks meet. This is similar to the way of how Gnutella and FreeNet networks can merge and evolve.

Also in distributed tree approaches replication is used in order to increase failure resilience. Replication can be simply achieved by having multiple peers sharing the same leaf of the search tree. Another form of replication is the replication of routing table entries. At the higher tree levels there exist always multiple peers that can serve as routing entries. By storing multiple, alternative references in the routing tables, in case of peer failures, there exists the possibility to use alternative routes.
P-Grid Construction Algorithm (Bootstrap)

When peers meet (randomly, e.g. random walk)
- Compare the current search paths p and q

Case 1: p and q are the same
- If split condition satisfied extend the paths, i.e. to p0 and q1 else replicate data

Case 2: p is a subpath of q, i.e. q = p0...
- If split condition satisfied extend the path p by the inverse, i.e. p1,

Case 3: only a common prefix exists
- Forward to one of the referenced peers
- Limit forwarding by recmax

The peers remember each other and exchange in addition references at all levels

Split conditions
- below a maximal path length
- storing a minimal number of data items

The interesting part about P-Grid is that it can be constructed in a completely decentralized fashion. Peers meet randomly (for example, by using ping messages or random walks), and when they meet they compare their current paths p and q. Three cases may occur. If they are same paths, the peers can decide to create a new split in the tree by extending their paths by 0 and 1 respectively. In the second case one path is a subpath of the other, thus one peer is more specialized than the other. Then the peer with the shorter path may decide to specialize opposite to the other peer, and delegate the responsibility for the keys in the "other half" to the peer with path q. Only in the third case, where the paths have only a common prefix (which includes the case that they are completely different if this prefix is empty) the peers cannot directly refine the search structure. Then they forward each other to other peers using their routing tables. In that way it is guaranteed that the peer eventually will meet a peer where case 1 or 2 occurs and a further refinement of the access structure is possible. Whenever peers meet they also exchange data items that belong to each others zone, and at each level they can exchange references from their routing tables for their matching paths.
Case 1
**Case 1**

peer 1 mutual exclusive case

peer 1 meet peer 2

peer 1 replicate peer 2
Load Balancing in P-Grid

- Split criterion: minimal number of data items
  - Each node has same storage load
  - Algorithm still converges quickly

If peers use as splitting criterion their current storage load, i.e. they only extend their path if a sufficient large number of data items is found that justifies such a decision, then the load is evenly balanced during the construction process (something that is not achieved with fixed node assignment, as for example in Chord). This is illustrated by this simulation result: the black bars indicate the data distribution, i.e. the percentage of data items that is associated with each of the 32 possible prefixes of keys at level 5 (or prefixes of keys), whereas the white bars indicate the number of peers that have chosen one of these prefixes for their path (therefore the tree will be unbalanced). One can see that the both distributions are closely correlated.
P-Grid Discussion

- **Performance**
  - Search latency: \( O(\log n) \) (with high probability, provable)
  - Message Bandwidth: \( O(\log n) \) (selective routing)
  - Storage cost: \( O(\log n) \) (routing table)
  - Update cost: low (like search)

- **Qualitative Criteria**
  - search predicates: prefix searches
  - global knowledge: key hashing
  - peer autonomy: peers can locally decide on their role (splitting decision)

With P-Grid we obtain performance characteristics that are equivalent to those of Chord. However, it exhibits a higher degree of self-organization due to its randomized construction process.
Example 2: Distributed Hash Tables (Chord)

- Hashing of search keys AND peer addresses on binary keys of length \( m \)
  - e.g. \( m=8 \), key("jingle-bells.mp3")=17, key(196.178.0.1)=3

- Data keys are stored at next larger node key

  peer with hashed identifier \( p \),
  data with hashed identifier \( k \), then
  \( k \in [ \text{predecessor}(p), p ] \)

Search possibilities
1. every peer knows every other
   \( O(n) \) routing table size
2. peers know successor
   \( O(n) \) search cost

In the following we introduce three approaches for constructing a resource location infrastructure, that are currently developed in research, but that are expected to be soon widely used on the Internet. Each of the approaches is based on a different abstract model in order to organize a distributed data access structure.

The first approach is based on the idea of distributing a hash table (Chord). Data and node identifiers are mapped into the same key space. We assume that the keys are arranged on a circle (or in other words all computations 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. Such an organization would lead to linear search cost or linear routing table size, when using a naïve approach to organizing the data access structure.
Routing Tables

- Every peer knows m peers with exponentially increasing distance

Each peer p stores a routing table. First peer with hashed identifier s_i such that 
s_i = \text{successor}(p + 2^i - 1) for i = 1, \ldots, m

We write also s_i = \text{finger}(i, p)

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 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 size.
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 it is responsible for the data key and the data has been found, or there exists such a peer and then the request is forwarded. 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).
A Chord network can be reorganized by joining and leaving of nodes. In such an event the peer who joins the network has to build up its own routing table. It can do this by repeatedly searching for the necessary entries in its routing table. 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 correspondingly, which can be done at the same cost.
The behavior of Chord has been analyzed by means of simulations. One important issue is whether the workload that each peer receives, in terms of data items being stored at the peer is uniformly distributed. In the situation depicted, with a uniform distribution, one would expect 50 keys per node. As we see there exist nodes with more than 450 keys and many with no keys. The problem is that the IP addresses do not map uniformly into the data key space.
The search performance is as expected very good. The length of the search paths is closely concentrated around \( \frac{1}{2} \log_2(n) \). The factor \( \frac{1}{2} \) is explained by the fact that the search starts, if we consider the routing tables as an embedding of search trees into the network, at a randomly selected tree depth.
Chord Discussion

- **Performance**
  - Search: like P-Grid
  - Node join/leave cost: $O(\log^2 n)$
  - Resilience to failures: replication to successor nodes

- **Qualitative Criteria**
  - search predicates: equality of keys only
  - global knowledge: key hashing, network origin
  - peer autonomy: nodes have by virtue of their address a specific role in the network

With respect to robustness, Chord can apply replication of data items. The data items are stored in that case at a fixed number of successor peers, such that when a peer fails, the data item can be located at its successor. Chord has a number of qualitative limitations: since it is based on a hashing approach the only search predicate that can be supported is key equality. The assignment of peers to their location in the key space is fixed by the global IP, therefore peers lack autonomy in deciding which data they want to store. The network can only evolve from a single origin, which implies that either an agreed upon entry point exists or a global identification needs to be maintained for a distinguished Chord network.
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 addresses of peers responsible for the neighboring zones for routing
  - Search requests are greedily forwarded to the peers in the closest zones

- Assignment of peers to zones depends on a random selection made by the peer

In topological routing a geometric space is used as key space, both for peer and data keys. The space is a d-dimensional torus, where the dimension d is usually low (e.g. 2-10). Peers are responsible for volumes in the space, which means they store data items with keys belonging to this volume. Search requests are routed to neighboring peers with coordinates, which are closer to the searched data key with respect to the geometry of the key space.

When peers join, they are free to select their peer key.
Network Search and Join

Node 7 joins the network by choosing a coordinate in the volume of 1

The first figure illustrates a CAN organization and search. One observes that the spaces can be divided into subvolumes of different sizes. When a search is performed, e.g. starting at coordinate (x,y), it is forwarded stepwise to closer peers, till the search arrives at the correct subvolume.

When a node joins the network it can decide for which subvolume it would like to support by selecting a coordinate. This is illustrate by the figure on the right.
Assume peer 7 decides for a point that lies in the subvolume that peer 1 is currently responsible for. First it performs a search for this point, finds out that peer 1 is responsible for the respective volume and splits the volume with peer 1. Each of the two nodes are then responsible for one half of this volume. The routing tables need to be updated for the neighborhoods of peer 7 and peer 1, which requires O(d) operations.
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 creates, for every data item $r$ replicas and reduces search time as there is a higher probability that the search starts already close to the target. The other possibility 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.
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 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).

It can be shown that the search complexity is $O(d \cdot n^{(1/d)})$ for $n$ nodes and dimension $d$. Thus 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 $O(\log n)$. 
CAN Discussion

• Performance
  - Search latency: $O(d^{n/d})$, depends on choice of $d$ (with high probability, provable)
  - Message Bandwidth: $O(d^{n/d})$, (selective routing)
  - Storage cost: $O(d)$ (routing table)
  - Update cost: low (like search)
  - Node join/leave cost: $O(d^{n/d})$
  - Resilience to failures: realities and overloading

• Qualitative Criteria
  - search predicates: spatial distance of multidimensional keys
  - global knowledge: key hashing, network origin
  - peer autonomy: nodes can decide on their position in the key space

An interesting aspect of CAN is that it is possible to exploit the "semantics" of the key space, to support more complex search predicates. For example, it would be possible to encode "semantic proximity" by properly encoding the data keys into the hashed key space. Another possibility would be to map a physical network topology into a CAN key space such that neighboring nodes in the CAN space are also physically close. This would reduce the search cost (in terms of search latency, or number of hops required in the underlying network).
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**

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>8e47683dda0932uje89</td>
<td>ZT38hwe0jh02hhdg2zu</td>
<td>tcp/125.45.12.56:6474</td>
</tr>
<tr>
<td>45e36we040dd00ked01</td>
<td>Rhweui2340jd901230</td>
<td>tcp/67.12.4.65:4711</td>
</tr>
<tr>
<td>f3682jink99dm4mxic</td>
<td>eqwe1089341h0qzhq3</td>
<td>tcp/127.156.78.20:8811</td>
</tr>
<tr>
<td>wen09hfh03uhm4218</td>
<td>erweg38382hjh3728ee7</td>
<td>tcp/78.6.6.7:2544</td>
</tr>
<tr>
<td>712345j89b8nopleldh</td>
<td>tcp/40.56.123.234:1111</td>
<td></td>
</tr>
<tr>
<td>d0ui43203803ujoeqjhh</td>
<td>tcp/128.121.89.12:9991</td>
<td></td>
</tr>
</tbody>
</table>

FreeNet has been developed as an advanced P2P file sharing system with the following goals:

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 approach to search.

In FreeNet each node stores some index information and in addition also some data related to the index information. For that purpose it maintains a cache as shown above. The size of the cache 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

FreeNet uses the cached index information for processing of search requests: if the data or the address is not directly found in the table (this would be comparable to Gnutella with caching!), then FreeNet forwards the search to only one peer (i.e. this is comparable to a random walker). However, the peer with the lexicographically closest key in the cache is chosen for forwarding. 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 should eventually improve the search latency.
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 Files

- First a the key of the 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 proceeds similarly as search. Only care has to be taken that the key of the new file does not already exist. Therefore the proposed key is first searched for and only if no collision is detected, the file 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.
This is a typical simulation result for FreeNet 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 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.
FreeNet Discussion

• Performance
  - Search latency: low (small world property)
  - Message Bandwidth: low (selective routing)
  - Storage cost: relatively low (experimentally not validated !)
  - Update cost: low (like search)
    • but a bootstrapping phase is required
  - Resilience to failures: good (high degree of replication of data and keys)

• Qualitative Criteria
  - search predicates: with encryption only equality of keys
  - global knowledge: none
  - peer autonomy: high (with encryption risk of storing undesired data)
## Comparison

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Search Type</th>
<th>Search Cost (messages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnutella</td>
<td>Breadth-first search on graph</td>
<td>String comparison</td>
</tr>
<tr>
<td>Freenet</td>
<td>Depth-first search on graph</td>
<td>Equality</td>
</tr>
<tr>
<td>Chord</td>
<td>Implicit binary search trees</td>
<td>Equality</td>
</tr>
<tr>
<td>CAN</td>
<td>d-dimensional space</td>
<td>Equality</td>
</tr>
<tr>
<td>P-Grid</td>
<td>Binary prefix trees</td>
<td>Prefix</td>
</tr>
</tbody>
</table>
Summary

The following questions apply to Chord, P-Grid, CAN and FreeNet:

- What is the expected search cost?
- How are other nodes affected when new nodes enter the network? What is the cost of node insertion?
- Which replication strategies are used?
- What global knowledge or central coordination is required?
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 (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
  - Requires the existence of short paths 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???
Random Graphs

- For many years typical explanation was - random graphs
  - Low diameter: expected distance between two nodes is $\log_k N$, where $k$ is the outdegree and $N$ the number of nodes
  - When pairs of 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.
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
- \( L \sim \frac{n}{2k} \)
Highly clustered
- \( C \sim \frac{3}{4} \)

Random Graph (k=4)
Short path length
- \( L \sim \log_{2}N \)
Almost no clustering
- \( C \sim \frac{k}{n} \)
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
Flashback: FreeNet Search Performance

- Modifying routing tables in FreeNet through caching has a "rewiring effect"
- Studies show that FreeNet graphs have small-world properties
- Explains improving search performance

Regular graph:
- \( n \) nodes, \( k \) nearest neighbors
- \( \Rightarrow \) path length \( \sim n/2k \)
- \( 4096/16 = 256 \)

Random graph:
- path length \( \sim \log(n)/\log(k) \)
- \(~ 4\)

Rewired graph (1% of nodes):
- path length \( \sim \) random graph
- clustering \( \sim \) regular graph

Small World Graph

A possible explanation of what is happening can be found in the graph structure of FreeNet. Recently a class of graphs has been characterized, that exhibits so-called small world characteristics. This was motivated by the observation that the distance of acquaintances in social networks is rather small on average (For example, a famous experiment performed by Milgram suggests that this distance is for the US 6, therefore this phenomenon is also called the 6 degrees of separation). An explanation for this phenomenon can be given as follows: acquaintances are normally local in nature. Therefore an acquaintance graph would typically have a very regular structure such as on the left (Note that this is the structure of the graph used for the initial network topology on the previous slide). This graph has the following properties: it is highly clustered, which means that the probability that two neighbors of a specific node are also connected with high probability. E.g. among the 4 neighbors of a node on the left hand side, only one pair is not connected. The average distance among nodes is however high.

On the other hand, from the theory of random graphs it is known, that the average path length is logarithmic in the number of nodes, but the clustering is very low. Such a graph is shown on the right. A low diameter (average path length) is however good for search, e.g. using flooding.

The interesting discovery was, that there exists a class of graphs which has the high clustering effect of regular graphs, but already has the property of short diameter of random graphs. Such a graph is shown in the middle. Essentially, it can be obtained, by introducing a few "short-cuts" randomly. One can think about people making visits to far away places, and thus introducing shortcuts in the graph of acquaintances. In Freenet this effect is obtained by the rewiring of graph during insertion and search.
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 in FreeNet?
P2P Overlay Networks as Graphs

- Each P2P system 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
  - Find a decentralized algorithm (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
Kleinberg’s Small-World Model

- Kleinberg’s Small-World’s model
  - Embed the graph into an r-dimensional grid
  - Constant number p of short range links (neighborhood)
  - q long range links: choose long-range links such that the probability to have a long range contact is proportional to 1/d^r

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

![Diagram showing Kleinberg’s Small-World Model with r = 2]
Influence of "r" (1)

• Each peer \( u \) has link to the peer \( v \) with probability proportional to
  \( \frac{1}{d(u,v)^r} \)
  where \( d(u,v) \) is the distance between \( u \) and \( v \).

• Optimal value: \( r = \text{dim} = \) dimension of the space
  • If \( r < \text{dim} \) we tend to choose more far away neighbors (decentralized
    algorithm can quickly approach the neighborhood of target, but then slows
de down till finally reaches target itself).
  • If \( r > \text{dim} \) we tend to choose more close neighbors (algorithm finds quickly
    target in it's neighborhood, but reaches it slowly if it is far away).
  • When \( r = 0 \) - 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

©2007/8, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis
P2P Systems - 48
Influence of “r” (2)

• Given node $u$ if we can partition the remaining peers into sets $A_1, A_2, A_3, \ldots, A_{\log N}$, where $A_i$ consists of all nodes whose distance from $u$ is between $2^i$ and $2^{i+1}$, $i = 0, \ldots, \log(N-1)$.
  - Then given $r = dim$ each long range contact of $u$ is nearly equally likely to belong to any of the sets $A_i$.
  - When $q = \log N$—on average each node will have a link in each set of $A_i$. 

![Diagram showing the partition of peers into sets $A_1, A_2, A_3, A_4$.](image)
Traditional DHTs and Kleinberg model

P-Grid's model

Kleinberg's model
Conclusions from Kleinberg's Model

- With respect to the Watts and Strogatz 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)
Summary

• How can we characterize P2P overlay networks such that we can study them using graph-theoretic approaches?

• What is the main difference between a random graph and a SW graph?

• What is the main difference between the Watts/Strogatz and the Kleinberg model?

• What is the relationship between structured overlay networks and small world graphs?

• What are possible variations of the small world graph model?
References


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


References

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


