Overlay and P2P Networks

Additional material on DHTs

Prof. Sasu Tarkoma

13.2.2014
Additional material

Butterfly networks and Viceroy

Skip graph

CANON: merging Chord rings

De Bruijn graph
Butterfly Geometry

A *k-ary n-fly* network consists of $k^n$ source nodes, $n$ stages of $k^{n-1}$ switches, and $k^n$ destination nodes. The network is unidirectional and the degree of each switching node is $2k$. The diameter of the network is logarithmic to the number of source nodes. At each level $l$, a switching node is connected to the identically numbered element at level $l + 1$ and to a switching node whose number differs from the current node only at the $l$th most significant bit. The main drawback of this structure is that there is only one path from a source to a destination, in other words, there is no path diversity. In addition, butterfly networks do not have as good locality properties as tori.
Butterfly network (with a tree)
Viceroy

The key point in Viceroy is the emphasis on **constant degrees**. The primary motivation was to develop an algorithm that has constant linkage cost, logarithmic path length, and best achievable congestion under the constraints.

It generally has constant degree such as CAN. Its degree is smaller than in Chord, Tapestry, and Pastry.

Viceroy assumes a **global ordering** on all the nodes in the system, which may make practical deployments in decentralized environments challenging.
Viceroy network

The idea is to approximate a butterfly network

The butterfly network results in constant node degree and thus state

The algorithm is rather involved

Idea is to use the butterfly levels for routing and then vicinity search

Message is routed upwards to the butterfly network root, and then downwards towards the correct destination, a shortcut may be used to reduce the routing cost
# Viceroy

<table>
<thead>
<tr>
<th>Viceroy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foundation</td>
</tr>
<tr>
<td>Butterfly network</td>
</tr>
<tr>
<td>Routing function</td>
</tr>
<tr>
<td>Routing using levels of tree, vicinity search</td>
</tr>
<tr>
<td>System parameters</td>
</tr>
<tr>
<td>Number of peers N</td>
</tr>
<tr>
<td>Routing performance</td>
</tr>
<tr>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Routing state</td>
</tr>
<tr>
<td>$Constant$</td>
</tr>
<tr>
<td>Joins/leaves</td>
</tr>
<tr>
<td>$\log N$ Note: assumes global ordering of nodes</td>
</tr>
</tbody>
</table>
A **skip graph** is a probabilistic structure based on the **skip list** data structure. The skip list has simple and easy insert and delete operations that do not require tree rearrangements. Thus the operations are fast.

The skip list is a set of **layered ordered linked lists**. All nodes are part of the bottom layer 0 list. Part of the nodes take part in the layer 1 with some fixed probability. For each layer there is a probability for a node to be part of that layer.

As a result the upper layers of a skip list are **sparse**. This means that a lookup can quickly go through the list by traversing the sparse upper layer until it is close to the target.
Skip graph II

The downside of this approach is that the sparse upper layer nodes are potential hotspots and single points of failure.

Skip graphs address this limitation and introduce multiple lists at each level to improve redundancy. Every node participates in one of the lists at each level.

On average $O(\log n)$ levels are needed in the structure, where $n$ is the number of nodes.
Skip Graph III

The skip graph is a distributed version of the skip list and its performance is comparable to the other DHTs.

Each node in a skip graph has average of log $n$ neighbours.

The main benefit of the structure comes from its ability to support prefix and proximity search operations. DHTs guarantee that a data can be located, but they do not typically guarantee where the data will be located.

Skip graphs are able to support location-sensitive name searches, because they use ordered lists.
CANON: Adding Hierarchy to DHTs

Most DHTs that have been proposed are flat and non-hierarchical structures. They thus contrasts the traditional distributed systems, which have employed hierarchy to achieve scalability.

A hierarchical DHT can be constructed that retains the homogeneity of load and functionality of the flat DHTs. A generic construction called Canon has been shown to offer the same routing state and routing hops trade-off found in the flat DHT designs.

The benefits of this approach include fault isolation, adaptation to the underlying physical network and its organizational boundaries, and hierarchical storage of content and access control.
The nodes keep their original links. Each node \( m \) in one ring creates a link to a node \( m' \) in the other ring if and only if:
- \( m' \) is the closest node that is at least distance \( 2k \) away for some \( 0 \leq k \leq N \)
- \( m' \) is closer to \( m \) than any node in the ring of \( m \)
De Bruijn Graph

An $n$-dimensional de Bruijn graph of $k$ symbols is a directed graph representing overlaps between sequences of symbols. It has $k^n$ vertices that represent all possible sequences of length $n$ of the given symbols.

In a $n$-dimensional de Bruijn graph with 2 symbols, there are $2^n$ nodes, each of which has a unique $n$-bit identifier.
Creating a de Bruijn graph

The node with identifier $i$ is connected to nodes $2i \mod 2^n$ and $2i + 1 \mod 2^n$.

A routing algorithm can route to any destination in $n$ hops by successively shifting in the bits of the destination identifier.

Routing a message from node $m$ to node $k$ is accomplished by taking the number $m$ and shifting in the bits of $k$ one at a time until the number has been replaced by $k$. 
De Bruijn Graph

Consider a node \( n \) with identifier \( b_1 b_2 \ldots b_k \), \( b_i \in \{0, 1\} \)

\( n \) has an out-edge to the nodes with identifier \( b_2 \ldots b_k 0 \) and \( b_2 \ldots b_k 1 \).

Node 00: out edge to 00 and 01
Node 01: out edge to 10 and 11
Node 10: out edge to 00 and 01
Node 11: out edge to 10 and 11

This adjacency scheme, based on shifting the identifier strings associated with a node yields a simple prefix based routing policy.
Constructing de Bruijn Graphs

De Bruijn graph for $2^m$ node network can be constructed in a recursive fashion from a $2^{m-1}$ node network.

Take the edge of the $2^{m-1}$ node network

Add a node in the middle

Details:
Example: Adding a digit

Example: Adding a digit