HyperCup *

Boyko Syarov
{syarovb@informatik.uni-freiburg.de}

Institut für Informatik,
Albert-Ludwigs-Universität Freiburg,
Freiburg im Breisgau, Germany

13.Mai.2007

1 Introduction

P2P networks are very popular today. Such networks are useful in many purposes - maybe best known for file-sharing. A pure p2p network doesn't have clients or servers, but only peers (peer nodes). They are connect to each other and act simultaneously as 'clients - servers'. In this network there is no central server managing the network, there is no central router. All peers act as equals. The P2P network itself relies on computing power at the ends of a connection rather than from within the network itself.

P2P networks can be categorized in categories:

- Centralized peer-to-peer network (Napster [4])
- Decentralized p2p network (Kazaa [5])
- Structured p2p network (Can, Chord [6] , HYPERCUP)
- Unstructured p2p network (Gnutella [2])
- Hybrid p2p network (JXTA [3])

I'll concentrate on the HyperCuP p2p network. Based on hypercube (n-dimensional cubes) this is a structured, well organized network. Because of the full graph structure and inefficient broadcast mechanisms most p2p networks do not scale well, limit the numbers of peers in the network and have unacceptable search times. To avoid these 'disadvantages' I'll provide efficient topology construction, algorithm, which reaches all the peers with minimum number of messages possible. I'll explain the hypercube network model and requirements in section 2. A complete algorithm for building

*Based on [1] Hypercubes, Ontologies and Efficient Search on P2P Networks

a 3-hypercup graph we be described in section 3. In this example we'll see basic features such as connecting additional peers, sending messages between them, disconnecting and 'cloning' peers. In section 4 I'll show an ontology-based network based on concept clusters and show that with more knowledge about the network we can improve its performance. And in section 5 I'll conclude.

2 Network model, requirements and organization in Hypercup

The hypercube has to be symmetric and balanced for most efficiency - means, we have to provide such structure, so that minimum messages are send to reach all the peers in the network. Peers which can communicate each other with directly send messages are called neighbors. Also all nodes in the network must have identical capabilities and tasks - there are no 'Superpeers', no node has a more prominent position than other. Every node should be able to be a root of a spanning tree in the network. The diameter of the network Δ - shortest path between most distant peers - is very important property for the search and broadcast algorithms. It has to be at most at logarithmic and each peer holds information of at most $log_2 n$ nodes. Another point of the hypercup model is that the topology has to provide redundancy - node failures must not disconnect the graph or make searching harder.

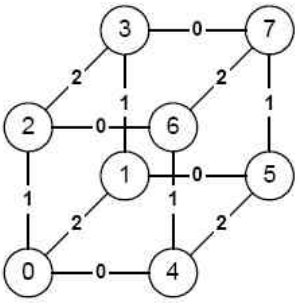

Figure 1: 3 - Hypercup graph

To make a balanced hypercube we need $N = b^{Lmax+1}$ nodes, where b is the base (in the examples I'll be dealing with dual base $b = 2$) and $(Lmax+1)$ - the number of dimensions. In Figure 1 [1] we can see a complete 3 dimensional Hypercube or 3-Hypercup. There are edges in the graph and those are labeled - node x is i-neighbor for node Y ($X = i(Y)$) if node X is in dimension i for Y. As example we can use Fig. 1 again. Node 6 is 1-neighbor for node node 4 and vice versa, means that edges are undirected. We can

[1] All images taken from [1]

also build extended neighbors $X = N(Y) = \{y0, y1, \ldots\}$, where N is termed neighbor link set, and represents the sequence of i-neighbors, X has to follow to reach Y and vice versa (i-neighbors should be unique). In our example in fig 1., the neighbor link set $\{1, 2\}$ leads from node 6 to node 5 and back., i.e. $6 = \{1, 2\}(5)$ and $5 = \{1, 2\}(6)$. The edge labels normally start at $i = 0$ and the maximum number is termed by $Lmax$ (dimensions - 1). Every node has its location in the graph topology described by two sets - a set of neighbor link sets: $\Omega = \{\{\}, N_1, N_2, ...N_n\}$ and a set of node addresses: $A = \{localhost, addr_1, addr_2, ..., addr_n\}$. It should be noted that when the hypercube is not completed, for example it has 9 nodes, the constructing algorithm clones some of the existing nodes and creates complete hypercube. This operation we'll see in the next section. For the search algorithm I'll suppose that the HyperCuP is completed.

2.1 Search and broadcast

HypercuP guarantees that during the forwarding process the set of node traversed increases - exactly once receive nodes a message. Or to reach all the nodes in the topology, there are $N - 1$ messages needed, which will be reached (the nodes) in $log_2 N$ steps. Let see how does it works: A node invoking broadcast sends a message to all of its neighbors, using the labeled edges. Node 'marks' on which label the message was sent. Nodes receiving messages 'allow' the forwarding of the message on these edge labels, which are higher than the marked. For example node 5 sends a broadcast - at first to all of his neighbors $\{1, 4, 7\}$. These nodes respectively forward the message to these links with higher edge number - node 1 to node 0 and 3, node 7 to node 6. In the third forwarding step node 3 send message to node 2, so all of the nodes can be reached. (Fig. 2). The search in the hypercube is broadcast with time to live, also broadcast with limits scope.

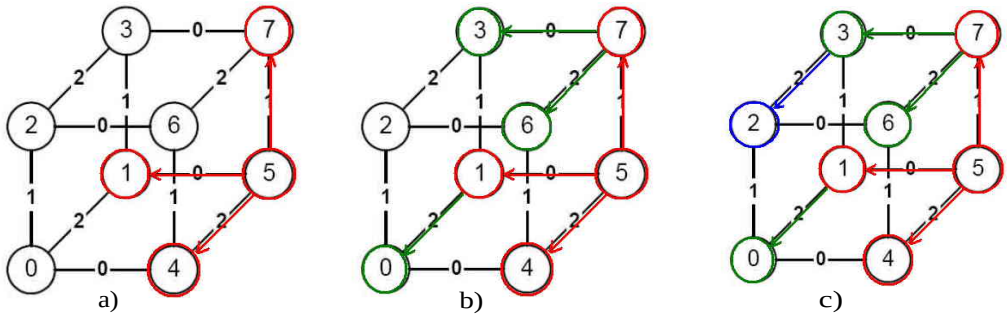

Figure 2: Forwarding steps a) 1-st step b) 2-nd step c) 3-th step

3 Building and Maintaining HyperCup

I'll build a complete 3-HC in following example, starting from one peer. Here you will see how the main construction algorithm works - joining, leaving and cloning procedures will be described. To have network symmetry, any peer should be allowed to accept and integrate new peers in the network. The idea is, that the hypercubes organized himself so, to have the topology of the next biggest complete hypercube graph - by covering the missing nodes with the existing and give this position up later to the new nodes that join the network. And we'll have a time of $O(log_b N)$ of messages had to be sent in order to join or leave the network. Cases were several peers want to join or leave the network at same time, won't be discussed in this paper.

At the beginning we have only peer 0 as active. In the next step peer 1, who wants to join the network, contacts to peer 0. A peer integrates a joining peer on its vacant position, and the neighbor levers are order such that lower levels come first. Peer 0 integrates peer 1 as its 0-neighbor, since it is alone in the network. Both peer established a link between them.

In the next step peer 2 wants to join the network. It can contact one of the two peers. Assume that it contacts peer 1. The 0-neighbor of the peer is already taken by peer 0, so is the first dimension also. Peer 1 opens a new dimension, because of peer 2. Now it becomes the integration control node for the network, and should completely integrate peer 2. It has to provide all necessary links to the peer, in order at the end peer 2 to carry out broadcasts. This includes all neighbor links connecting the existing neighbor levels (in this case only peer 0 and 1). So peer 1 has 0- and 1-neighbors and it has to provide peer 2 also with those neighbors. It will integrate itself as 1-neighbor, and since there are no other nodes in the network peer 0 becomes a temporary 0-neighbor for peer 2.(Fig 3).

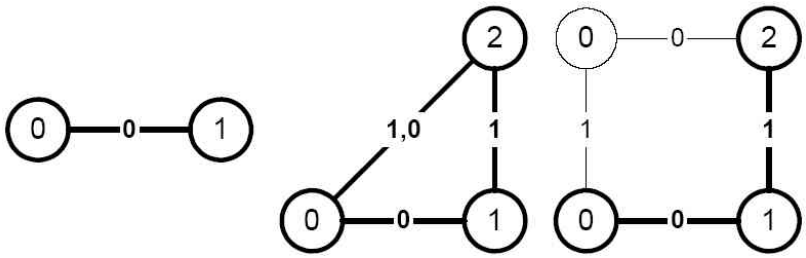

Figure 3: Building the hypercup

So peer 0 covers a vacant position, acts as it occupies two positions, but that is only temporary, because it cannot have more than one 0-neighbor. As you can see the thin line represents the link with the 'cloned' peer 0. And the relationship between peers 2 and 0 is tagged with the link set $\{0, 1\}$

instead of {0}. The link describes the path from peer 0 to peer 2. These temporary link sets are always constructed by this rule.

In next step peer 3 connects the network. There are three possible cases: peer 3 connects peer 0, 1 or 2 in order to join the network.
Case 1: It connects peer 0 to join, so peer 0 has to provide all the necessary links to peer 3. The new peer occupies the first vacant neighbor level - level 1 in this case. It will cover the temporary position that peer 0 has as a 0-neighbor for peer 2. Peer 0 also passes his link set $\{0, 1\}$ to the new peer and knows now, that the path to peer 2 leads through peer 3. At the end peer 3 has established link set {0} to peer 2 and link set {1} to peer 0. as you can see in fig. 4a.

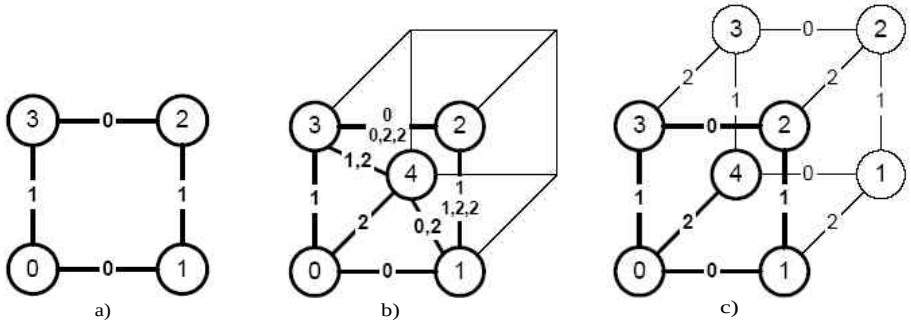

Figure 4: a) Integrated peer 3 b) Peer 4 c) Peer 4 temp. links

Case 2: If peer 3 joins the network due peer 2, peer 2 integrates it as its 0 neighbor. (because this neighbor level is free). It spreads the information to peer 0, who maintained the integration control. Note that [from Mario schlosser p.6]. As peer 0 becomes the information it does the same as in case 1, so that at the end peer 3 has a 0- and 1- neighbor.

Case 3: This case is the worst-case for network. In order to connect to new peer 3, peer 1 should open a new dimension, since 0- and 1-neighbors are already occupied. This will lead to very unbalanced structure with a lot of temporary peers and hampering search. But the graph will be misbalanced only if there are no joining peers. Normally, the information about a vacant position in network is always spread, so that the joining peers could connect peers with vacant neighbor, to cover the gap, and balance the structure.

New peer 4 arrives and to join. It contacts peer 0. Since the 0- and 1-neighbor levels are occupied, peer 0 open new dimension for the peer. It will be intergraded as 2-neighbor for peer 0. The new peer should have also 3 neighbors, but since its alone in the dimension, the nodes in the network, should cover these positions. The closest peers to the vacant position are chosen to cover it. If there is more than one peer, the next on the highest

neighbor level should cover the position. So peer 1 will be temporary 0-neighbor, peer 3 - 1-neighbor. Now only 0- and 1-neighbors for peers 3 and 1 are missing. This position will be taken by peer 2. For complete view, see fig. 4b,4c (thin links and nodes means temporary peers). We should manage also the link sets: As temporary node, peers 3 and 1 have the link set $\{1, 2\}$ and $\{0, 2\}$ to the new node. There exists also a second link set from node 3 to node 2. This is when we go through temporary peers 3 and 2. So the link set is given by $\{2, 0, 2\}$ (or in ascending order $\{0, 2, 2\}$). The same is with peers 1 and 2. There is a path through peers 1 and 2 and the link set is also $\{1, 2, 2\}$.

In the next step let us assume that peer 0 wants to leave the network. This peer decides which existing peer will cover his position. In this case peer 0 leaves only one vacant position, so it will find one other peer to cover it. If there were more positions occupied by peer 0, it will have to find more successors for its responsibilities. Also, as before, the closest peer is chosen - peer 4. This peer establish a link sets with peer 3 $\{\{1, 2\}, \{1, 2\}\}$ and with peer 1 $\{\{0, 2\}, \{0, 2\}\}$. The new graph with the node responsibilities can be seen in Fig. 5.

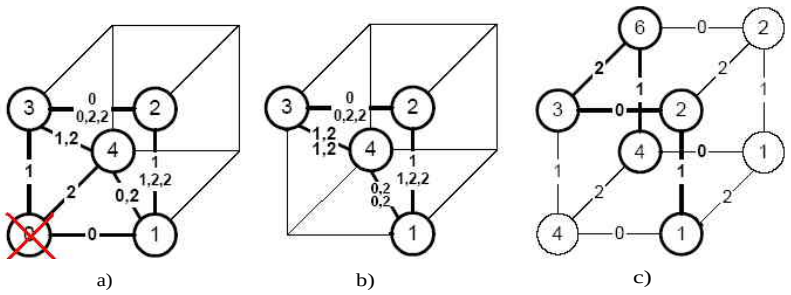

Figure 5: a) Peer 0 leaves the network b) Network state c) Network state with temp. links

We continue to add more peers in the network until we get a complete 3-hypercup. The next step is when a new peer 5 contacts peer 1 in order to join. Peer 1 still doesn't have a 2-neighbor (its cloning covers the vacant position), so it integrates the new peer as its 2-neighbor. The link sets are also updated - now peer 4 has a 0-neighbor - namely peer 5 and peer 1 can get rid of its $\{\{1\}, \{1, 2, 2\}\}$ link set, replacing it with $\{1\}$. Still the links between node 5 and 2 are missing, but it will be set to $\{0, 2\}$. Check Fig. 6 for the updated graph structure.

Next, peer 6 wants to join the network. It contacts peer 3, who has the integration control. Peer 3 has a temporary 2-neighbor peer 3 (itself), so peer 6 will be placed there. All the links will be passed to the new link. Now

peers 6 and 2 share the link set $\{0, 2\}$. See fig 6 for the current hypercube state. I should mention that it is regardless of which peer contacts peer 6, because all of the peers share a temporary node in the network. So peer 6 will be integrated without misbalancing the hypercube. (Fig. 7a)

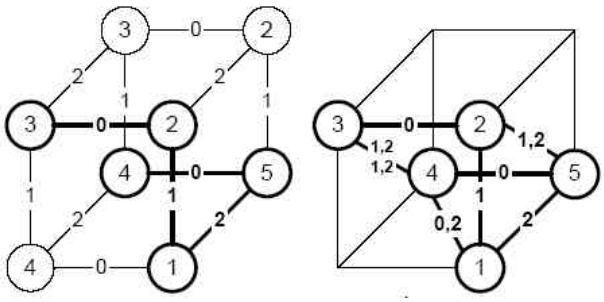

Figure 6: Peer 5 joins the network

When peer 7 contacts peer 6 to join the network, we are almost done with the competed 3-hypecube. Peer 6 has a temporary 0-neighbor - peer 2, so it forwarded the integration control to that peer. Peer 2 replace its cloning und updates the link sets, so that at the end of the integration peer 7 is the new 0-neighbor for peer 6, 2-neighbor for peer 2 and 1-neighbor for peer 5. (Fig. 7b)

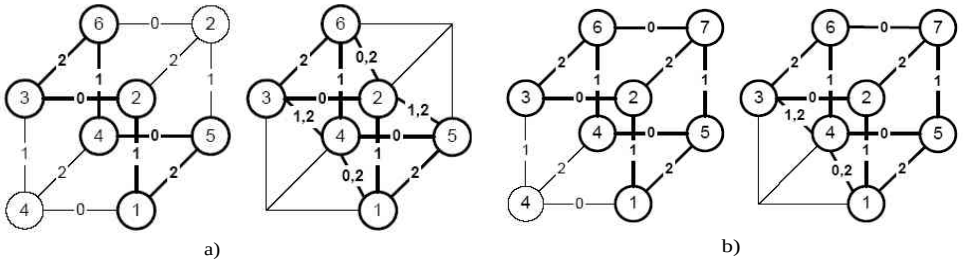

Figure 7: a) Peer 6 joins the network b) Peer 7 joins the network

The final step is when peer 8 contacts either to node 3, 4 or 1. All these three node share the temporary node 4. The links will be updated and the complete 3-hypercube graph can be seen (Fig. 8). If there are link failures, there are disconnected peers from the graph, but the network topology should be able to recover the graph. The algorithm contacts the disjoined nodes neighbors and asks them for their own neighbors, so that the departure step should be carried out, means which node will replace the missing one.

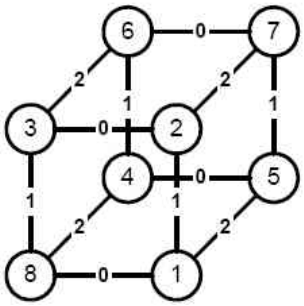

Figure 8: Complete 3-Hypercube

4 Ontologies and concept clusters

The hypercup networks scales well have very good search time. With some additional knowledge for the network, we can improve the performance. Peers hold information, and we can organize them into concept - clusters, so that only those clusters will receive message, which can probably give a right answer. These concept clusters can be organized in one big ontology, which represents the relationship between them. And for every cluster we use separate hypercube graph. As example of ontology-based network we can take a music server. We can see we have a hypercube for uploading, downloading and separate hypercube for music genres(Fig. 9). Note that we can have the same peers in more that one hypercube, because of the information they shared.

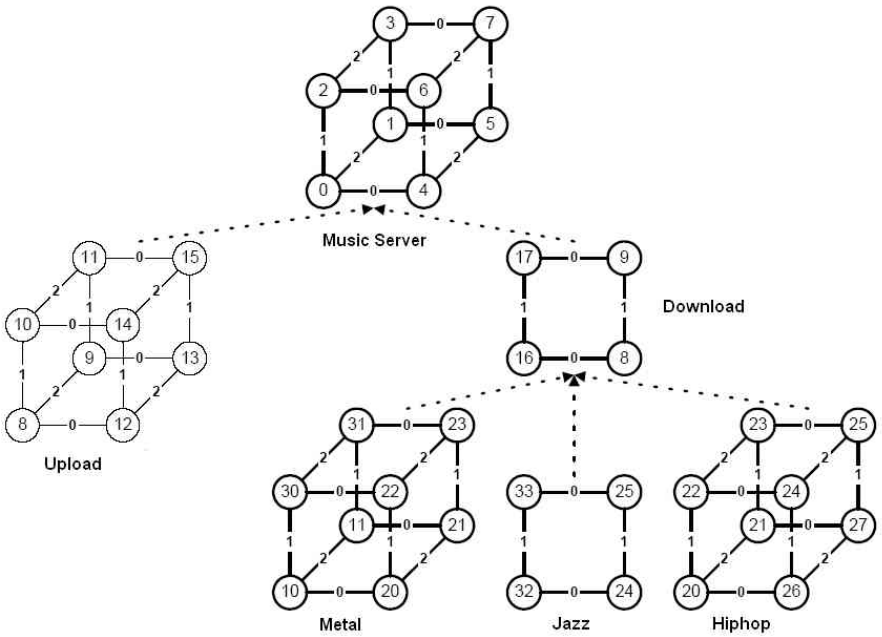

Figure 9: Ontology-stuctured

5 Conclusion

Hypercube can be seen as very good p2p network, because of its efficient broadcast and search algorithm. This network doesn't requires supernodes or central servers - it can organize itself. We have a logarithmical diameter and small number of messages is sent to update the network status. Peers, who share common information can build concept clusters, which will improve the search time. This can be useful in domains, where we can search for specific categorizes only from the specific cluster and from the entire network.

References

[1] M. Schlosser, M. Sintek, St. Decker, W. Neijdl *http://infolab.stanford.edu/∼schloss/hypercup/*

[2] Gnutella website: http://www.gnutella.com

[3] JXTA website: http://www.jxta.org.

[4] Napster website: http://en.wikipedia.org/Napster

[5] Kazaa website http://www.kazaa.com

[6] I. Stoica et al. Chord: A scalable p2p lookup service for internet applications. In *Proc. ACM SIGCOMM01*, 2001.