Informatics Research Proposal:
Range Queries in Peer-to-Peer Systems

Christina Kaskoura
s0679008
C.Kaskoura@sms.ed.ac.uk

1. Motivation

Peer-to-peer systems (P2P) are distributed systems which allow the sharing of resources in a decentralized and autonomous way. The nodes of the system (peers) are all equal and can join or leave the system whenever they want, thus making the peer-to-peer network extremely dynamic. Since there is no one node or set of nodes responsible for routing and communication between peers, this is done through a self-organizing overlay network which runs on top of the physical network. Given the increased number of applications that are built over peer-to-peer networks, the development of such overlay structures that perform efficiently certain desired tasks is becoming a very important and interesting problem. In this project we concentrate on such overlay networks that facilitate the location of data in the peer-to-peer network.

During the past few years, a big part of the research on peer-to-peer networks has concentrated on the issue of efficiently locating a value in a peer-to-peer system. This means that we need to determine which peer holds this value and retrieve it. A number of different solutions have been proposed for this problem, the most popular of which are based on a structure called a Distributed Hash Table (DHT). In DHTs the identifiers of the nodes participating in the peer-to-peer system are hashed as well as the keys of the data that is to be stored in the system. Then, the data is distributed among the nodes depending on their hash values and the hash values of the nodes, while algorithms are provided to determine in which node each key is (or should be) stored depending on its hash value. This way, the data and thus the load balance is evenly distributed among the peers.

Different approaches to this problem mainly vary on the way they choose to distribute the data among the nodes. Chord for example [SMK+01] organizes the nodes in a modulo 2^m ring where m the number of bits in the hashed ids of the keys and the nodes and assigns each key to the first node whose id is equal or follows the id of the key. Each node in the ring has pointers to the nodes that succeed it by 2^i where $1 \leq i \leq m$ and this way any value can be located efficiently in O(logn) steps. Pastry [RD01] on the other hand assigns each key to the node that is numerically closest to it. Routing is done by forwarding any query for a particular key at each step to a node that shares with the key a prefix that is at least b bits longer than that of the current node or one that is numerically closer to the key than the current node.

Unfortunately, as it can be seen, the approaches that are based on DHTs cannot efficiently answer range queries since the hashing involved does not preserve information on the initial ordering of the data. This turns the development of an overlay network that can efficiently answer such queries into a very interesting problem since as it can be seen from traditional databases range queries constitute a big percentage of the queries usually issued to a database. Certain solutions to this problem that still use DHTs have been proposed but they are not efficient enough since they only give approximate answers to the range queries or require a priori division of the values into partitions assigned to different peers which can obviously

be a problem given the dynamic nature of peer-to-peer networks and even if applied it can easily lead to uneven load balance on the peers.

Therefore, a few other solutions that do not use DHTs have been proposed to address this problem. In this case, another challenge is to balance the load throughout the network, since there is no hash function to guarantee that the data will be evenly split among the peers. One such solution is the use of P-Trees [CLG+04] which are basically B+-trees distributed among the peers so that no one node is responsible for answering to the queries and thus overloaded. However, this approach assumes that each node has predefined data and thus does not deal at all with the partition and distribution of the data. Moreover, it assumes that each node only holds one tuple and although it can be extended with "virtual peers" to support multiple tuples in each node it does not scale to large amounts of data.

In [GBG04] a solution to the load balancing problem is proposed that can be applied to parallel databases as well as peer-to-peer systems. The authors develop and analyse algorithms that ensure that all the nodes of a parallel database have an even balance while they also study and demonstrate their use in peer-to-peer systems. Although a skip graph data structure for the peers is proposed, the range partitioning and maintenance of an even load balance between the nodes are examined independent of an underlying peer-to-peer network and more emphasis is given on these tasks than on the development of a specific overlay network.

Finally, [JOV05] propose a binary balanced tree structure as an overlay network for peer-to-peer systems that can answer range queries efficiently, called BATON. Each peer in the network keeps links to its parent, children, adjacent nodes and selected nodes in the level of the tree in which it is situated, which are used for the efficient routing of queries. Algorithms for answering queries as well as for nodes joining and leaving the network are provided and a technique for adjusting the network in order to preserve an even load balance is also developed.

From what is mentioned above, it can be seen that there are not many available solutions that efficiently address the problem of range queries in peer-to-peer networks. Even in the case of BATON which is one of the best proposals so far, it is necessary for the peers to keep a lot of routing information and exchange an increased number of messages in order to answer to queries or allow nodes to join or leave the network. Therefore, it seems that there is still a big research field available as far as this problem is concerned, since different approaches may yield even better results.

In this project, we will study such a different approach and develop and evaluate an overlay network for peer-to-peer systems that will be able to efficiently answer range queries. As it was argued before, such a project has the potential to contribute to the development of the research field presented and is of great theoretical as well as practical interest.

2. An initial approach

We present a naïve algorithm that can construct an overlay network for a peer-to-peer system to efficiently answer range queries. In the presentation that follows we assume that the range of values we want to distribute among the peers is from 0 to x but the algorithm is the same for any other bounds.

At first, only one node a is present in the network and so this node is responsible for the entire range of values. When a second node b decides to enter the network, it will have to contact node a, who will assign to this node half of the values it is responsible for. Each node will keep a data structure showing the value range it is

responsible for as well as the node or nodes that are responsible for the rest of the values, the way it is presented in Figure 1.

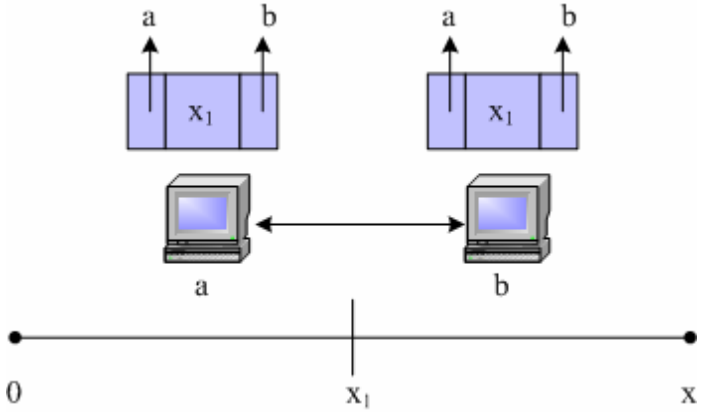

Figure 1

Assuming now that a third node c, who knows of b, wants to enter the network, it will contact b, which will assign to it half of the values it is responsible for and so the network will change to what is depicted in Figure 2. Here it must be noted that although node c is aware of the presence of node a since it got this information from node b, node a is not aware of the presence of node c and so it will route all the queries referring to the values c is responsible for through b. The routes that can be followed in the network are shown as arrows in the figure.

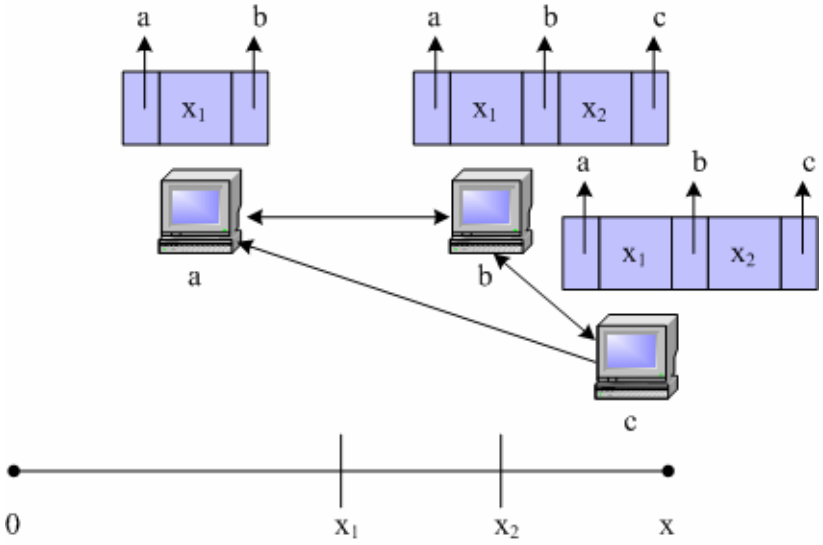

Figure 2

The same way, each node that wants to enter the network will contact another node of the network, which will assign a range of values to it and give it the routing information it holds. Therefore the insertion of a new node into the network can be done in only one step. In order for a node to leave the network it will have to contact a node that is responsible for a range of values adjacent to its and make it responsible for its range of values as well. Furthermore, all the nodes that have an entry to their routing table corresponding to the node which is leaving will have to update their routing information. In the worst case all the nodes in the network will have information on the node which is leaving, as it is the case for node a, and thus O(n) steps are needed for a node to leave the network.

Using the algorithm described earlier to insert new nodes into the network, we can represent the distribution of the values to the nodes as a binary tree, the way it is shown in Figure 3. The internal nodes of the tree hold the values used for the division

of the domain, while the leaves of the tree hold the nodes which are responsible for the various ranges that result from this partitioning.

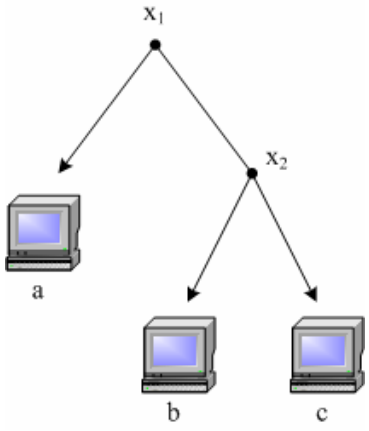

Figure 3

After a number of insertions, this binary tree should ideally be balanced, as it is shown in Figure 4. This way, given the fact that each node gets its routing information from the node it contacted in order to join the network the nodes in the right subtree will not be aware of any node in the left subtree but node a and vice versa. The same applies for all the smaller subtrees contained in the binary tree.

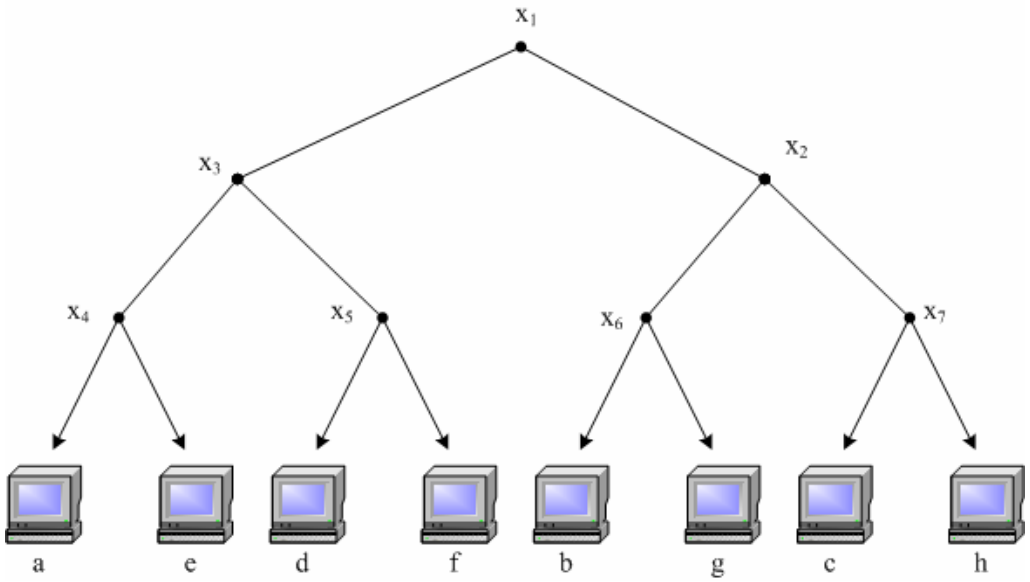

Figure 4

Thus, if node h, for example, receives a query for a range of values for which node f is responsible, it will route this query through node a, which in turn will route it through node d, which will finally send it to node f. This amounts to the query being first routed to a node in the range 0-x_1, then a node in the range x_3- x_1 and finally to node f which is responsible for the range x_5- x_1. This is shown in Figure 5 with thick arrows. This in turn means that in this ideal case any query can be answered in at most O(h) steps, where h the height of the tree, and since in a balanced binary tree $h = \log_2 2n = \log_2 n + 1$, where n the number of leaves in the tree which is the number of nodes in the network, we can answer any query in at most O($\log_2$n) steps. The problem in this case is that since the nodes on the right subtree have no knowledge of the nodes in the left subtree except for node a, any queries originating from the right subtree that refer to a range for which a node in the left subtree is responsible will

have to go through node a and the same goes for queries originating in the left subtree which will have to go through node b.

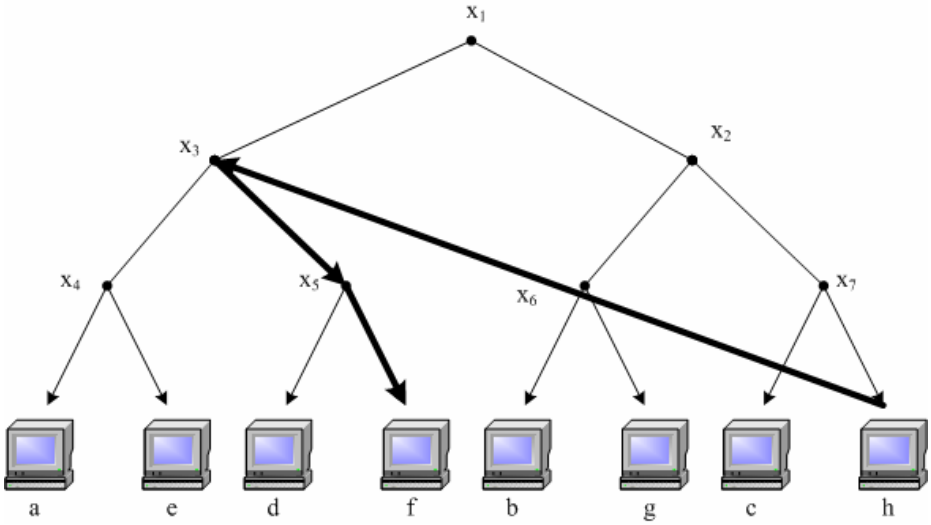

Figure 5

Furthermore, in the worst case the tree will not be balanced but will have the form depicted in Figure 6. In this case we can still answer any query in at most O(h) steps, but this time O(h)=O(n). What is worse, in this case node a is responsible for half the values in the initial range, while node e is responsible for only 1/16 of them.

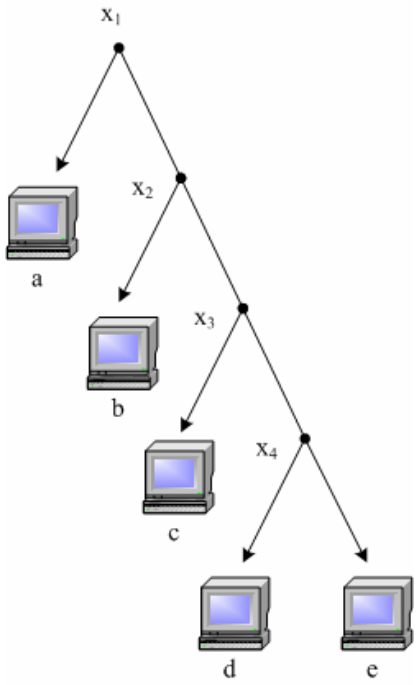

Figure 6

Therefore, this algorithm needs to be improved so as to divide the values equally among the nodes and keep the binary tree that comes from this division balanced. Furthermore, the issue of the hot spots present here has to be addressed so that there is no one node or set of nodes through which all queries must go, while finally the algorithm will have to be expanded in order to also handle involuntary node departures, which have not been discussed here. This improvement and extension of the algorithm is what will present the bulk of the work in this project.

3. Implementation

As part of this project we will implement a simulation of an overlay peer-to-peer network that follows the principles described in the previous sections. This simulation will be implemented in Java 1.5 and will include all the possible interactions between the nodes of the network. These interactions are basically the following:

Join: A node asks another node to insert it into the network the way it was described in the previous section.

Assign (range): A node assigns part of (or all) the values it is responsible for to another node.

Inform (routing table): A node gives another node the routing information on the network it currently holds in its routing table.

Delete: A node informs another node that it is leaving the network.

Search (range): A node asks another node for a range of values it is searching. The node asked will return any of these values it is responsible for and forward the query for the rest of the values (if any) to the node which is responsible for them according to its own routing table.

Return (values): A node returns a set of values it received a query for as it is described in the previous interaction.

Therefore, a node of the network may be represented in terms of implementation the way it is presented below.

Node
Routing table
Join Assign (range) Inform (routing table) Delete Search (range) Return (values)

Of course, any other interactions that might come up after the expansion of the algorithm we presented in the previous section will also be included in the simulation.

4. Goals of the project

As it was mentioned earlier, in this project we are developing an overlay network for a peer-to-peer system which will be able to efficiently handle range queries. Therefore, for the project to be considered successful the proposed network will have to fulfill the set of requirements presented here.

First of all, by use of the mechanisms and methods provided by the proposed overlay network we should be able to answer range or exact match queries in logarithmic time. This means that the network should be able to locate the node or nodes on which a value or range of values resides in a number of steps or messages exchanged among nodes logarithmic in the number of nodes in the network. Moreover, there should be no one node or set of nodes through which the majority of the queries have to go in order to be answered thus creating hotspots in the network.

Furthermore, the network should be able to handle nodes joining or leaving the network, whether they are leaving voluntarily or involuntarily. This should also be done in a number of steps or messages logarithmic in the size of the network, both in the case of nodes joining the network as well as in the case of nodes leaving it.

Finally, the network should keep a balanced load among the peers, in the sense that no peer should be responsible for more than twice the number of data objects any other peer is responsible for. In this point we assume that there is no significant change in the set of values distributed among the peers and so when the network is in a steady state (no peers are joining or leaving) there is no need of rearranging the values to keep a balanced load. Therefore, the only rearrangement the network needs to do to preserve the load balance is when nodes join or leave the system, as described earlier.

References

[AAB+05] L.G.Alex Sung, N.Ahmed, R.Blance, H.Li, M.A.Soliman and D.Hadaller, *A survey of Data Management in Peer-to-Peer Systems*, In Web Data Management, Winter 2005

[CLG+04] A.Crainiceanu, P.Linga, J.Gehrke and J.Shanmugasundaram, *Querying peer-to-peer networks using P-Trees*, In Proc. WebDB, 2004

[GBG04] P.Ganesan, M.Bawa and H.Garcia-Molina, *On-line balancing of range-partitioned data with applications to peer-to-peer systems*, In Proc. VLDB, 2004

[JOV05] H.V.Jagadish, B.C.Ooi and Q.H.Vu, *BATON: A Balanced Tree Structure for Peer-to-Peer Networks*, In Proc. VLDB, 2005

[RD01] A.Rowstron and P.Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, In Proc. Middleware, 2001

[SMK+01] I.Stoica, R.Morrism D.Karger, M.F.Kaashoek and H.Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, In Proc. SIGCOMM, 2001