Plaxton Routing

From a Peer-to-Peer-Network Point of View

Lars P. Wederhake

21.9.2008

Contents

1 Abstract

Think of a heterogeneous network to which millions of people are connected, say the today's Internet, and now contemplate the people's strongest desire: Sharing information, say files, with others - since humans are social beings. But given such a distributed network similar files tend to turn up at several places randomly distributed over the whole network. Peers now claim the wish to request a desired file within the shortest time possible, if shared at all, but meanwhile regarding an efficient usage of network resources.
Plaxton Routing provides a general idea of achieving this goal and therefore became of great interest in later P2P-Networks, such that a quite similar procedure is used in Pastry and Tapestry. This paper's main intent is to provide the reader an easily accessible introduction to the general concept of Plaxton Routingas well as supplying additional motivation to study further towards this research area.

2 Introduction

The first generation of peer-to-peer networks like Napster actually represented a client-server structure. Quite soon after the advent of the new idea of sharing information followed the first peer-to-peer networks like GNUtella that eventually deserved their name. This network was extremely flexible due to the decentralized structure but lacked performance and scalability. Moreover, there was no certainty that a file request returned with a hit if there existed a copy of the desired object. Finally, the third generation of these networks fixed this drawback of GNUtella et al. All known approaches made use of somewhat similar to a Distributed Hashtable (DHT). Interesting about some of these overlay networks is that those make intensive use of Plaxton Routingwhich was already developed in 1997 with no idea of P2P-networks. Self-evidently, during the late 90s the rapid development of the the Internet forced people to consider more efficient access schemes to shared objects. And indeed several articles about sharing memory in distributed environments had been published.[CGP97]. Plaxton et al. , however, designed a randomized algorithm for fast access to copies of some objects (e.g. files of any kind) exploiting locality in a quite elegant way. Their research emphasized to reduce costs for an individual access to an object, for inserting and deleting a copy of an object A, for taking off the net a node or adding one and finally the amount of additional memory to store routing information on a single node.
Nevertheless, it was necessary to limit their research to a certain class of cost function which will be introduced later on. Section 3 stresses this issue describing the necessary abstractions and restrictions while Section 4 depicts the allowed operations in a Plaxton mesh in detail. Finally, in Section 5 the analysis gives evidence of the claimed results.

3 Modelling the Real World

Describing an efficient distributed data structure taking all factors into account means a truly complex and extremely challenging task. Only to provide an impression which parameters might play a role: edge delays, edge capacities, buffer space, communication overhead or patterns of user communication. In order to face this complexity Plaxton et al. have focused on a simplified model that makes certain assumption e.g the network topology. But this simplification causes some advantages such as a simpler design of practical implementations of the algorithm.

3.1 Communication

The Plaxton Mesh - is modeled as a complete graph $G = (V, V^2)$ with $n = |V|$ nodes each sharing a set $\boldsymbol{A}$of m = poly(n) objects. Every node can be identified by log(n)-bit string. That's why Plaxton Routingexpects the underlying network to be able of establishing reliable connections between any two nodes in G. Since the main goal of this procedure is to provide an efficient access to copies of a certain object A which is given when the costs of communication is low. But before minimizing these costs it is necessary to formulate such a function. Let $c : V^2 \rightarrow \mathbb{R}$ denote the cost for transferring a single word from a node u to a node v in G. The previously declared cost function is symmetric and fulfills the triangle inequality such that, (i) c(u,v) = c(v,u) and $c(u, \omega) \leq c(u, v) + c(v, \omega)$ for any u, v, ω . For convenience, it is assumed that for u, v, ω : $c(u, v) = c(u, \omega)$ iff $v = \omega$. As messages often consist of more than one word and cost do not necessarily rise linear the authors additionally introduced a non-decreasing function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ such that f(1) = 1. Accordingly, the total costs of transmitting a message of the length l from node u to v are $f(l(A)) \cdot c(u, v)$.
In order to proove the correctness of the model Plaxton Routingis restricted to a certain class of cost functions. It is claimed that there are two constants δ, Δ such that for every node u in V the ratio of neighbors within a given radius r and 2r varies at most among those constants. This might be an assumption which is likely correct for most networks including the fat-tree but whether this is also correct for heterogeneous networks like the Internet which includes peer-to-peer networks is questionable. A formal definition of this issue might be described as the following: let M(u,r) define the ball around $u \in V$ with radius $r \in \mathbb{R}^+$ such that $M(u, r) = \{v \in V : c(u, v) \leq r\}$. Then should be valid: $min\{\delta|M(u, r)|, n\} \leq |M(u, 2r)| \leq \{\Delta|M(u, r)|\}$

The authors of Plaxton Routingdecided to only support the reading operation since the ability to write addresses the issue of how to keep consistency and cover security affairs.
Let u an v be nodes in G and u stores a copy of an object A in $\boldsymbol{A}$. The node v might access this iff u has declared this object as shared. If not it is in status unshared and cannot be found and therewith not accessed from any v in G.
Plaxton Routingfinally offers three operations which can be executed: Read, Insert, Delete. The Read operation requests an objects, the Insert operation makes a copy of an object A in $\boldsymbol{A}$accessible and the Delete operation undoes this.

3.2 Objects

Objects in this model are any kind of information a node can share with others independent of the status it is in. In the real world this might be any file: A jpg, mp3, mpg, divx file or perhaps something more random like streams as it is used in P2Live etc. Important, however, is that for any object A in $\boldsymbol{A}$there is log(m)-bit identification. This e.g. might be a hash key of the filename. Actually this could be a simple but bad idea since the same data can be shared under a different name. This is why it is suggestive to construct a cryptographic hash key of the whole file. Furthermore, throughout this document A^i is used to denote the i-th bit in $log(m)$-identification. The size of a file is referenced by a mapping l :$\mathbf{A}\rightarrow \mathbb{N}$ which in fact is a $\mathcal{O}(logn)$-bit string.

3.3 Local Overhead

Every efficient data structure needs a certain overhead of information in order to administer the workflow. It is just a matter of finding a good balance between both dimensions - memory and time. It is clear that if on every node there is copy of an object A that the access is tremendously fast but consequently the memory overhead is enormously large. The other opposite is that no local information is stored - which means absolutely no overhead - but then it might be necessary to search the whole network for a copy.
Apart from these both extremes Plaxton Routingprefers a compromise which will be explained in 3.2.1. Therefore it is clever to divide the local memory in so called main memory and in auxiliary memory. The main memory stores local copies of shared objects of $\boldsymbol{A}$. The auxiliary memory, however, serves as a storage for the routing information, say that information which is needed to speed up the process of accessing copies in the network. The ratio of both main memory on the one hand and auxiliary memory on the other gives a good indicator of the efficiency of the routing protocol. Additionally, one might consider the computation overhead as every node has got to administer its routing table but actually this does not produce any especially complex operations and therefore the authors of Plaxton Routingdid not calculated this in their model.

3.4 Addressing and Routing Scheme

In the run-up of describing the routing mechanism there several definitions to state to map the real world to a mathematical model. Let $[n]$ denote the set $\{0, \dots n-1\}$ while for $x \in V$ x^i represents the i-th bit of the identification string of node x. Additionally, it is assumed that $\exists k : |V| = n \leq (2^b)^k$ for some positive constant b. If there are more nodes there will be two nodes with the same identification which causes Plaxton Routingto fail. Therefore k is fixed and must be carefully chosen - especially as the same address space is taken for the mapping of objects. Moreover, $x[i]$ defines $x^{(i+1)b-1} \dots x^{ib}$ for any i in $log(n)/b$. Likewise $A[i]$ references $A^{(i+1)b-1} \dots A^{ib}$ that is the $i+1$-th character (resp. bit segment of length b).
It appears reasonable to provide an example at this place. Given a network with b =

4. Now one can reference each node with a 16-bit string but as $2^4 = 16$ it is possible to reference each node with a string of four hex signs. E.g. $x = F4A0$, or $y = 0C3E$. Then is it simple to reference each sign in the above described way like x[2] = 4. Finally, it is obligatory to define a total order to the nodes. This is accomplished by defining the following bijection $\beta : V \rightarrow [n]$.

3.4.1 An informal description of the routing mechanism

Plaxton Routingis as its name already suggests a routing mechanism. This mechanism achieves the fast access by exploiting locality and forwarding the request where required(distributing control information).A quite similar procedure exists in real life. Even the best student will someday be told to solve an exercise which seems too difficult to solve until the next day but thankfully we all have our peers, our network, so out there might be people who know an approach and if not we know someone, who in turn knows someone who knows it. Plaxton Routing, however, extends this scheme by so called shortcuts and takes proximity into account. Transferred to the real life example it means that before asking a co-worker at the faculty one better ask someone who had to solve the problem earlier in time, e.g. last semester, and that's why she owns a sample solution which in fact symbolizes the so called shortcuts exploiting proximity. But Plaxton Routingcan do even better. It for example guarantees to find a copy of an object A if there is one at all. Before we explain how this is accomplished it is necessary to clarify two small but decisive things.

Neighbor Lists In order to manage the requests with regard to locality it is crucial to store information about the proximity relationship to other nodes. Therefore one part of the axillary memory is used to keep that information in a routing table. Before describing the issue more formal providing a general idea of the concept might be helpful. Every node in V and likewise every object A in $\boldsymbol{A}$owns a unique $\mathcal{O}(log(n))$ identification bit-string (for this paper we grant that no collisions will occur) distributed by a given hash function. Now bit segments are grouped to, say characters, and we obtain a denominational number system. Now it is likely that in a huge network the hash keys are approx. uniformly distributed and it seems obvious that within a given radius r more than one possible neighbor can be found. That is the reason why primary and secondary neighbors are handled individually which means there is one neighbor if there such a neighbor at all with shortest distance to this node; it is called the primary neighbor - all others are called secondary neighbors. But who are these neighbors and how they are related? Each node owns a routing table with rows and columns. A (i,j)-neighbor, say v of u, denotes the nearest neighbor with the same prefix and $u[i] = j$ ($\nexists \omega : c(u, \omega) < c(u, v) \wedge \forall k : u[k] = v[k], k \in [i] \wedge u[i] = j$).In order to limit the outer links, the amount of secondary neighbors is bounded by a certain constant d. The predicate for defining the group of (i,j)-neighbors of a certain node u is given by $\{v : \forall k : u[k] = v[k], k \in [i] \wedge u[i] = j \wedge c(u, v) \leq d * (c, \omega)$ where ω is the primary neighbor of u$\} = W_{i,j}$. If there is no such node then $W_{i,j} = \emptyset$. Those secondary neighbors which are stored in the routing table are $U = \{min\{d, |W_{i,j}|\}\}$. It's plausible just to use the nodes of U with smallest c(u,x). Finally, each node manages a set of reverse outbound links. These are called revers neighbors. Reverse neighbors of a given node v are all those nodes to which v is the primary (i,j)-neighbor.

Pointer List Besides the routing table with the links to primary and secondary neighbors. Each nodes stores a list containing links to objects. Following these leads directly to the place of the object's nearest location relative to the node who stores that shortcut.Actually, all shortcuts of a node are stored as triples in a list $Ptr(x)$. Triples contain an entry of the specific object, say A, the address of the node x which shares this object and an upper bound of costs $k = c(x, y)$. (A, x, k). It is guaranteed that at any point of time there is only one entry related to a certain object. Entries of the pointer list are generated through the insert command of a node. Evidently, the delete operation removes relevant entries.

Sequences

4 The Operations

In this section the basic operations will be explained in detail. First, the algorithm swill be described informally and later on they are specified through a formal set of instructions from which on the analysis in section 6 can be done. Basically, there are three operations Read, Insert and Delete.

4.1 Read

Given a fully functioning network with all entries of files etc correctly entered. Let u be a node who starts a request for a file A. As u starts the request it passes along the sequence $< u >$ of primary neighbors the current upper bound k. This happens equally sending it from $u_i - 1$ to u. But how does u_i proceeds? Actually, it might occur one of the following two possibilities. If u_i is the root for the object A, it will have an entry in it pointer list associated with A. This information is sent to u_0 to establish a connection between u_o and u_i Otherwise there is no file shared in the network. If u_i is not the root, it will browse both primary and secondary A(i,A[i])- neighbor(s) for a pointer link to A for an entry like $(A, x, k^{'})$. If $k^{'} \leq k$ a connection between both ends will be arranged by a request. Given the case this fails, u_i forwards the read request to u_{i+1} updating the upper bound k.

Action of x_i on receiving a message Read (x, k, y): -

If $i > 0$ and $x_i[i-1] \neq A[i-1]$, or $i = (logn)/b - 1$ (that is, x_i is the root for A) then:

•Node x_i sends a message Satisfy(x) to node v such that $(A, v, \cdot)$ is in $Ptr(x_i)$, requesting it to send a copy of A to x. If $Ptr(x_i)$ has no such entry, then there are no shared copies of A

Otherwise:

•Let U be the set of secondary $(i, A[i])$-neighbors of x_i . Node x_i requests a copy of A with associated upper bound at most k from each node in $U \cup \{x_{i+1}\}$.

•Each node u in $U \cup \{x_{i+1}\}$ responds to the request message received from x_i as follows: if there exists an entry (A, v, q_v) in $Ptr(u)$ and if $q_v^{'} = q_v + c(x_i, u) + \sum_{j=0}^{i-1} c(x_j, x_{j+1})$ is at most k, then u sends a success message Success(v, q_v) to x_i .

•Let $U^{'}$ be the set of nodes u from which x_i receives a response message Success (u, k_u). If $U^{'}$ is not empty, then x_i updates (k, y) to be (k_z, z), where z is a node with minimum k_u over all u in $U^{'}$.

•If $k = \mathcal{O}(\sum_{j=0}^{i-1} c(x_j, x_{j+1}))$ then x_i sends a message Satisfy(x) to node y, requesting y to send a copy of A to x. Otherwise, x_i forwards a message Read (x, k, y) to x_{i+1} .

4.2 Insert

Consider an insert request for an object A originating at a node u. Now two situations might occur. First, the object A has never been inserted or there was a copy of A in G but it was deleted at some point of time. That's why there is no shared copy, either. It obvious that one would not found an entry in pointer list. (Given the algorithm of the deletion procedure works, but we take this for granted). Alternatively, there is already one copy online and due to the described routing system one will meet at the latest at the root an entry in the pointer list indicating that copy. Interestingly, this insert operation works independently of both described cases which is great because it reduces complexity.
On attempting to insert the object A u sends an update request along the primary neighbor list $\langle u \rangle$. As soon as an insert message arrives at a node y_i it updates its pointer list. If an entry for the object does not exist so far it is inserted. Otherwise the old entry is updated if the new upper bound $(\sum_{j=0}^{i-1} c(j_i, j_{j+1}))$ on the cost of getting an object A from node u is smaller than the current one associated with the object. If it is smaller u_i passes the insertion request to u_i unless u_i is already the root. In any other cases the request stops here an the instruction processing terminates.

Action of y_i on receiving a message Insert (y, k): -

If $(A, \cdot, \cdot)$ is not in $Ptr(y_i)$, or $(A, \cdot, k_0)$ is in $Ptr(y_i)$ and $k_0 > k$, then:

•Node y_i accordingly creates or replaces the entry associated with A in $Ptr(y_i)$ by inserting (A, y, k) into this list.

•If $y_i[i-1] = A[i-1]$ then y_i sends a message Insert $(y, k+c(y_i, y_{i+1}))$ to y_{i+1} .

4.3 Delete

The deletion command eventually removes all entries in the pointer lists of the type $(A, u, \cdot)$ if A is the object which u does not longer want to share. Similarly to the insert operation the process delete(u) start at u_0 at passes the request down(towards the root) the primary neighbor sequence. When the request arrives at u_i it checks whether it holds a related entry. If this is true u_i behaves as the u_o and forwards the request to its primary neighbor according to A unless u_i is already the root for A. Granted that u_i does not own an entry associated with A u_i will not forward the delete request because now it is clear that there is already a copy of A nearer to u_i so that all even more distant nodes, relative to u_0 store pointer to other copies. But after a node on the sequence remove the entry it afterwards will ask its reverse neighbors (i-1, A[i-1]) whether they store a pointer in their lists. Now it might occur that more than one returns with an acknowledgement. Let X denote the set of nodes which returned with a Success - message. u_i will then select the node ω storing the triple (A, v, q_v). $\nexists x \in X : k_x + c(u_i, c_x) < k_\omega + c(u_i, \omega)$ where k denotes the costs.

Action of y_i on receiving a message Delete (y): -

If $(A, y, \cdot)$ is in $Ptr(y_i)$, then:

•Let U be the set of reverse $(i-1, A[i-1])$-neighbors of y_i . Node yi removes $(A, y, \cdot)$ from $Ptr(y_i)$, and requests a copy of A from each u in U.

•Each u in U responds to the request message from y_i by sending a message Success $(v, q_v + c(y_i, u))$ to y_i iff (A, v, q_v) is in $Ptr(u)$.

•Let U 0 be the set of nodes u such that y_i receives a message Success (u, k_u) in response to the request message it sent. If $|U'| > 0$ then y_i inserts (A, w, k_w) into $Ptr(yi)$, where w is the node in U' such that k_w k_u , for all u in U 0 .

•If y_i $i[i-1] = A[i-1]$ then y_i sends a message Delete (y) to y_{i+1}.

5 Analysis

In this section, then, the results which means the complexity of the described operations will be presented and eventually analyzed. As this paper's main intent is to provide a rough idea of how Plaxton Routingworks and give some performance hints as well as characterizing the greatest benefits and drawbacks. Consequently, this paper will make it short with an in-depth analysis. In order to get a fine grasp it is highly recommend to consult the original technical report by Plaxton and his fellows[CGP97]. First, elementary lemmas are going to be introduced but which do not necessarily will be proved as already stated. This will especially be done, if a lemma requires further lemmas to be proved. Alternatively, a draft or idea of how the proof might work will be given if it seems necessary to the understanding of the entire theorem's proof. By these lemmas, then, the four main theorems will be proved. Finally, the numbering of the theorems and lemmas remains similar to that one which is used in the original report. This, however, causes gaps in the numbering but eases up finding the related hypothesis and proof, respectively, in the technical report by Plaxton et al.

Constants Throughout the analysis and already in the model several constants turn up which have not been specified, yet and We will not do anymore in this paper because the relationships are complex since several equalities must be valid at a time. Additionally, as for the following analysis it is just necessary to know that can be adjusted appropriately and a few mostly general relations. Constants used are $b, d, \delta, \Delta, \gamma$ and ϵ with $d, \gamma << 2^b$ and $\epsilon < (1/10 \cdot 2^{b \cdot log_\delta 2})$

5.1 Communication

Lemma 5.9 In order to show that the costs for reading an object are asymptotically optimal is obligatory to assert that radius by taking a multiple of 2 more nodes in a ball around x does only increase by a certain constant.

Before demonstrating that it is necessary to declare some definitions:

- The node which starts the request is specified by x, the node which shares a copy is y.
- γ : an integer constant - chosen appropriately (cf. Constants)
- X_i^j resp. Y_i^j, where $i, j \in \mathbb{N}$. These variable specify the ball $N(x, \gamma^j 2^{(i+1)b})$ (resp., $N(y, \gamma^j \cdot 2^{(i+1)b})$).
- $X_{i^*}^1$ and $Y_{i^*}^1$ denote the ball Let i denote the least integer such that the radius of X_i^1 is at least $c(x, y)$
- a_i and b_i are referring to the radius of$X_i^1 (resp., Y_i^1)$.

Lemma 5.9: $\forall i \in [(logn)/b - 2]$: $2^{b \cdot log_\Delta 2} \cdot a_i \leq a_{i+1} \leq 2^{b \cdot log_\delta 2} \cdot a_i$ and $2^{b \cdot log_\Delta 2} \cdot b_i \leq b_{i+1} \leq 2^{b \cdot log_\delta 2} \cdot b_i$. For $i = (logn)/b - 2$: $a_{i+1} \leq 2^{b \cdot log_\delta 2} \cdot a_i$ and $b_{i+1} \leq 2^{b \cdot log_\delta 2} \cdot b_i$. Then a_i and b_i are both $\mathcal{O}(c(x, y))$.

Proof: Note: all which is valid for X is valid for Y (a and b likewise) as well unless otherwise stated. $\gamma << 2^b \rightarrow \forall i \in [(logn)/b - 2] \rightarrow |X_{i+1}^1| = 2^b \cdot |X_i^1|$ (For Y the same) Therefore, $\forall i \in [(logn)/b - 2]$, it follows from Equation 1 that $2^{b \cdot log_{Delta} 2} \cdot a_i \leq a_{i+1} 2^{b \cdot log_\delta 2} \cdot a_i$

For $i = (logn)/b - 2$, $|X^1_{i+1}| \leq 2^b \cdot |X^1_i|$ and hence, $a_{i+1} \leq 2^{b \cdot log_\delta 2} \cdot a_i$. If $i^* > 0$, then $a^*_i \leq 2^{b \cdot log_\delta 2} \cdot c(x, y)$. Otherwise, $a_i \in \mathcal{O}(2^{b \cdot log_\delta 2}) = \mathcal{O}(c(x, y))$.

Lemma 5.22 $\forall i \in [(logn)/b - 1] \wedge j \in \mathbb{N}$ $\Pr[\tau_i \geq j] \leq (10\epsilon)^j$. This lemma is reverted to several other lemmas dealing with probabilities of certain traits of random walks in a special directed graph which are not described here. Therefore, it is strongly recommended to consult the original report if there is deeper interest. For this paper, however, we assume that the hypothesis is correct.

Lemma 5.23 $\forall i \in [(logn)/b - 1]$: $E[c(x_i; x_{i+1})]$ and $E[c(y_i, y_{(i+1)})]$ are both $\in \mathcal{O}(a_i)$. Similarly to Lemma 5.22 this one makes intensive use of assumptions about random walks with which we, due to the brevity of this paper, do not want to deal with.

Proof of Theorem 1: Lemmas 5.9, 5.22, and 5.23 are used to establish Theorem 1.

- τ actually specifies a positive integer with several conditions to valid at a time. For this proof, however, is not compulsory to describe the interrelations in detail. It is sufficient to say that τ here denotes the steps necessary to locate a copy.
- By the definition of the alogrithms follows: $\mathcal{O}(\sum_{0 \leq i < \tau}(d^2 \cdot c(x_i, x_{i+1}) + c(y_i, y_{i+1})))$
- $\rightarrow f(l(A)) \sum_{i \leq i < \tau} \mathcal{O}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))$.
- Split sum in : 1) $E[\sum_{0 \leq i < i^*}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$
  2) $E[\sum_{i^* \leq i < \tau}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$.
- By Lemma. 5.9 and 5.19, we obtain as first term $= \mathcal{O}(a_{i^*} + b_{i^*})$.

  Finally, we provide about on expected costs for deleting an object: $E[\sum_{0 \leq i < i^*}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$:

  - Since $\tau = i^* + i$, by L. 5.18: $j \geq 0$, $Pr[\tau \geq i^* + j] \leq (10 \cdot \epsilon)^j$.
  - $\rightarrow E[\sum_{0 \leq i < i^*}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))] \leq j \cdot (10\epsilon)^j \cdot (a_{i^*+j} + b_{i^*+j})$
  - $\rightarrow j \cdot (10\epsilon)^j \cdot 2^{j \cdot b \cdot log_\delta 2}(a^*_i + b^*_i)$
  - $\rightarrow = \mathcal{O}(a_i + b_i)$ since $10\epsilon 2^{b \cdot log_\delta 2} < 1$. By L. 5.9.

5.2 Costs on Data related Operations

Lemma 5.6 Lemma 5.6 Let u be in V and let i $\in [(logn)/b]$. Then, the number of nodes of which u is an i-th level primary neighbor is $O(logn)$ whp. Also, $E[a_u] = \mathcal{O}(logn)$ and $a_u \in \mathcal{O}(log^2 n)$ whp.

Corollary 5.6.1 For any u in V , the total number of reverse neighbors of u is $\mathcal{O}(log^2n)$ whp, and expected $\mathcal{O}(logn)$.

We get by the probabilty that u is the (i,j)-primary root neighbor ($n \cdot e^{-n/2^{(i+1)b}}$ (cf. Lemma 5.5) a Pareto-distribution because $\leq n \cdot e^{\omega \cdot log(n)} = \mathcal{O}(1/poly(n))$. By the Chernoff-bound we then have: number of root prim-neighbors $\in mathcalO(log(n))$

Lemma 5.7 Let u belong to V , and let i be in $[(logn)/b]$. Then the number of i-leaves of u is $\mathcal{O}(2^{ib}logn)$ whp.

- We will proove $u \in V, i \in [log(n)/b] \rightarrow Numberofi - leaves(u) = \mathcal{O}(2^{ib} * log(n))$
- $v \in i - leaf(u) \rightarrow v \in N(u, c_0 \cdot 2^{ib}log(n)); c \in R$
- Corol. 1 $\forall j \in [i], v_j \in N(v_{j+1}, c_1 \cdot 2^{j+1}log(n))$
- Proof by Induction:
- Hypothesis: $\forall j \in [i+1], v = v_0 \in N(v_j, c_0 \cdot 2^{jb}log(n))$ whp
- Induction base j=0 trivial
- Induction step :
  - Assumption: $v \in N(v_j, c_0 \cdot 2^{jb}log(n))$
  - By Corol 5.5.1 : $v_j \in N(v_{j+1}, c_0 \cdot 2^{ib}log(n)$
  - remember if $v \in N(u, k_0) and \omega \in N(v, k_1) \rightarrow w \in N(u, \Delta^2 k_0 + \Delta k_1)$ but set $u = v_{j+1}$, $v = v_j$ and $\omega = v_{j-1}$
  - $\rightarrow v \in N(v_{j+1}, (\Delta^2 k_0 + \Delta k_1) \cdot 2^{jb} * log(n)$

Proof of Theorem 2: **Proof of Theorem 2:** Insert(A,x): The expected cost of the operation is bounded by $E[\sum_{i<(logn)/b} c(x_i, x_{i+1}] = \mathcal{O}(a_{(logn)/b-1}) = O(C)$.,(by Lemmas 5.9 and 5.23) Delete(y): By Lemma 5.6, $\forall i$:, the number of reverse (i,j)-neighbors of x_i for any j is $O(logn)$ whp, where x_i is the i-th node in the primary neighbor sequence of x. Therefore, the expected cost of the delete operation executed by x is bounded by the product of $E[\sum_{i<(logn)/b} c(x_i, x_{i+1})]$ and $O(logn)$. By Lemma 5.23, it follows that the expected cost of a delete operation is $\mathcal{O}(Clogn)$.

5.3 Auxiliary Memory

Proof of Theorem 3: Proof of Theorem 3: The first step is to place an upper bound on the size of the neighbor table of any u in V: The number of primary and secondary neighbors of u is at most $(d+1) \cdot 2^b(logn)/b \in \mathcal{O}(logn)$. The number of reverse neighbors of u is $\mathcal{O}(log^2n)$ whp (By Corollary 5.6.1). Finally, there is an upper bound on the size of the pointer list of any u in V. . The size of Ptr(u) is at most the number of triples of the form $(A, v, \cdot)$, where A is in $\mathcal{A}$ and v is in V such that:

- (i) $\exists i \in [(logn)/b]$ such that v is an i-leaf of u,
- (ii) $A[j] = u[j] \forall j \in [i]$
- (iii) A is in the main memory of v .

By Lemma 5.7, the number of i-leaves of u is $\mathcal{O}(2^{ib} logn)$ whp. $Pr[A[j] = u[j]] \leq 1/2^{ib} \forall j \in [i]$. Since the number of objects in the main memory of any node is at most l, it follows that whp, $|Ptr(u)| \leq \sum_{i \in [(logn)/b]} \mathcal{O}(l \cdot logn) = \mathcal{O}(l \cdot log^2 n)$. Combining the bounds on the sizes of the neighbor table and pointer list the size of the auxiliary memory of u is $\mathcal{O}(l \cdot log^2 n)$ whp.

5.4 Adaptability

Proof of Theorem 4: By Lemma 5.6, $\forall u \in U$, the number of nodes of which u is a primary or secondary neighbor is $\mathcal{O}(logn)$ expected and $\mathcal{O}(log^2 n)$ whp. Moreover, u is a reverse neighbor of $\mathcal{O}(logn)$ nodes since u has O(log n) primary neighbors. Therefore, the adaptability of the scheme is $\mathcal{O}(logn)$ expected and $\mathcal{O}(log^2 n)$ whp.

6 Summary

6.1 Results and Consequences

Plaxton Routingdescribes high-performant distributed data structures which exploits locality. Moreover as there is no central directory as the routing is calculated on local information. This multi-hop routing mechanism is based on an application level routing system and represents a quite fault-tolerant structure as there is always more than one path possible. Unfortunately, the designs turns out to be not well suited for dynamic node join and departure which makes this approach less interesting for peer to peer nets. Moreover, as in Plaxton Routingthe root is the last instance which might keep a pointer to an object it is the weakest point. This causes two shortcomings. First, doing maintenance on the root possibly has results in rejected requests for an object though the file is stored in the net. That is why Plaxton Routing' reliability is just to be considered moderate.

6.2 Open Questions & Critics

After all it is still not very clear whether the internet can modeled like the way G.C. Plaxton and his colleagues did. The consequence viz. is that the number of secondary neighbors might increase rapidly.[CS08]
Furthermore, the fact dynamic node join and departure is not supported remains in unacceptable, if Plaxton Routingshould be applied on a peer-to-peer net. Adapting this mechanism to this demand, however, seems to be quite difficult task which in fact would complicate the mechanism. Since there are often extremely high (or low) dimensioned

nodes (consider dedicated lines versus 56k modems) it might be a good idea to consider load-balancing in Plaxton Routing. And quite similar issue is the fact that some objects are many times more often requested than others. How could Plaxton Routingbe improved to fix this problem. Another question which is worth being asked is can a a geographic layout compete with proximity plus neighbor routing and if so how do nodes obtain the global knowledge?

7 Bibtex

References

[CGP97] Andrea W. Richa C. Greg Plaxton, Rajmohan Rajaraman. *Accessing Nearby Copies of Replicated Objects in a Distributed Environment.* 1997.

[CS08] Peter Mahlmann Christian Schindelhauer. *Peer-to-peer Netzwerke.* Springer, 2008.