PASTRY

R. HIMMELMANN

ABSTRACT. Pastry provides a versatile fault-tolerant and efficient framework for developing various kinds of peer to peer networks.

1. Network Structure

1.1. **Overview.** Pastry is a distributed hash table. The computers which are participating in the network are called *peers* or *nodes*. They are connected through a network called underlying network. This is usually the internet with TCP/IP. Pastry provides methods that allow peers to exchange messages in an efficient manner and computers not yet participating in the overlay to join it. Using these two methods applications can be developed.

1.2. **Proximity.** We assume that messages can be sent from a peer p to a peer q if p has sufficient information about q, for example its IP-address. We assume further that the cost for sending such a message may be measured by a metric d. That means especially $d(p, q) = d(q, p)$ and $d(p, q) \leq d(p, p') + d(p', q)$ and that this cost can easily be calculated. As an example for TCP/IP we would use the round trip time.

When we say that two nodes p and q are "near" we mean that $d(p, q)$ is small.

1.3. **What do we want to optimise?** Every operation should need as few messages as possible. If we still have the choice between multiple routes for a message we try to find one where the nodes are close to each other. Focusing primarily on the number of nodes for a message also has good implications for stability and security: If there are less nodes on the path of a message the probability that one of them is faulty is smaller.

We will see that good results concerning proximity by using a greedy algorithm: We first define an algorithm for routing which will have a certain degree of freedom for finding the next node for routing. With this freedom we will choose a node that is near to the current node. We will see that this can be achieved by filling the routing tables of nodes with nearby nodes.

1.4. **IDs.** IDs are numbers $\in \mathbb{N}/(2^{128}\mathbb{N}) \equiv \{0, ..., 2^{128}\} := ID$. We define a norm $|.|$ on ID trough $\forall b \in ID.|b| := min(b, 2^{128} - b)$. Every peer has an ID $id(p) \in ID$. These IDs are randomly generated or computed as the value of a cryptographic hash-function on the node's public key, if a node has a public key. This way we can assume that IDs are randomly distributed through the underlying network.

Every piece of data d also has an ID $id(d)$ and is associated with the peer p such that $|id(p) - id(d)| = min$. What exactly is meant by "a piece of data" depends on the application that is run on pastry.

We interpret IDs as numbers with base $|B| = 2^b$. Also we set $m = \log_{|B|} |ID|$. We define a function $pfxl : ID \to ID$ that which computes the longest prefix between two IDs written as strings: For two numbers $x = x_0 \circ x_1 \circ ... \circ x_{m-1}$ and $y = y_0 \circ y_1 \circ ... \circ y_{m-1}$ where $\circ$ is the concatenation $pfxl(x, y)$ is the i with $\forall j < i + 1 : x_j = y_j$ and $x_i \neq y_i$. For convenience we set $pfxl(p, q) := pfxl(id(p), id(q))$ for nodes p and q.

1.5. **The Routing Table.** Let p be a node. The routing table $R_p =: R$ of p is a matrix $(R[i, j])_{0 \leq i < m, 0 \leq j < |B|}$. Here $R[i, j] =: q$ is a node with $pfxl(p, q) = i$ and $id(q)[i] = j$. If there is no such node in the network (that we now of) or if $j = id(p)[i]$ we set $R[i, j] = null$. If possible we choose a q near to p. A schematic example of R_p with $id(p) = 1232$ and $|B| = 4$ follows:

	..1..	..2..	..3..	..4..
xxxx	*null*	2xxx	3xxx	4xxx
1xxx	11xx	*null*	13xx	14xx
12xx	121x	122x	*null*	124x
123x	1231	*null*	1233	1234

Theorem 1.1. *R_p for node p contains $\leq \mathcal{O}(\frac{\log n}{b} 2^b)$ entries.*

Proof. First we observe that the probability that there is another node q in the network with $pfxl(p, q) \geq m$ is $(n-1) * (2^{-b})^m = \mathcal{O}(n * 2^{-bm})$. m digits of $id(q)$ need to be the same as for $id(p)$ for each of which there are 2^b possible values. We now set $m = (c+2)\frac{\log n}{b}$ for a constant $c > 0$. Now the probability can be rewritten as $n * 2^{-bm} = n * 2^{-(c+2)\log n} = n * n^{-c-2} = n^{-c-1}$ which is small for large n. Therefore it is probable that $a_{i,j} = null$ for $i > (c+2)\frac{\log n}{b}$. □

1.6. **The Leaf Set.** The leaf set L for a node p is an array with $|L|/2$ nodes with next higher IDs and $|L|/2$ nodes with next lower IDs. If $|L|/2 > n$ nodes may be on both sides of the leaf-"set". This will always happen when the network is initialised.

1.7. **The Neighbourhood Set.** The neighbourhood set M of a node p contains $|M| = 2^b$ nodes near p. It is used for repairs and insertion of peers. It is not strictly necessary for the operation of pastry as there are implementations that do not use it at all.

2. Operations

2.1. **Routing.** We now present the algorithm for routing in pastry:

```
Search($r$)
if ($id(L[-|L|/2]) \leq r \leq id(L[|L|/2])$)
    Route to peer $p' \in L$, so that $|r - id(p)'|$is minimal.
    return;
$c \leftarrow pfxl(r, id(p))$
if ($R[c, r[c]] \neq null$)
    Route to peer $R[c, r[c]]$
    return
Route to a $p' \in R \cup L \cup M$ with
    $pfxl(r, id(p')) \geq c$ and
    $|r - id(p')| < |r - id(p)|$
```

Usually messages are first routed with the routing table and if that is not possible anymore the leaf set is used. The third part of the algorithm, routing to any node that decreases the distance to the destination, is used especially if there have been recent node failures.

The routing table is used to make routing fast. With it nodes may be selected in such a way that the distance to the target decreases exponentially, thus making routing in logarithmic time possible. The leaf set on the other hand is used for maintaining reliability in the network. The following proofs will illustrate this. In both we assume that all entries in the leaf sets and routing tables are filled with appropriate nodes, if possible.

Theorem 2.1. *Routing takes no more than $\mathcal{O}(n/|L|)$ steps.*

Proof. We only use the leaf set. Let $p_0, p_1, ..., p_{m+1}$ be the nodes along the route. For all but the last step we will use $L[|L|/2]$ or $L[-|L|/2]$. On average $|id(L_{p_i}[\pm|L|/2]) - id(p_i)|$ is $|ID|/n * |L|/2$ because the average distance between the IDs of two nodes with consecutive IDs is $|ID|/n$. With $|id(p_0) - id(p_{m+1})| \leq \frac{|ID|}{2}$ and a division we get the required value. □

Note that messages can always be routed if there is at least one node alive on each side of the leaf set of every node.

Theorem 2.2. *The expected value is $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ messages.*

Proof. We now use the routing table. Each time a p_{i+1} is found in the routing table $pfxl(id(p_{i+1}), r) > pfxl(id(p_i), r)$. After each step we have to look one row deeper into the routing table. With high probability only the first $\mathcal{O}(\frac{\log n}{b})$ rows in the routing table of any node are filled. So we will, with high probability, only make $\mathcal{O}(\frac{\log n}{b})$ steps in the routing table. After that the final node is in the leaf set with a probability of 0.98. The probability that no more than two steps in the leaf set are needed is 0.9994. This last part we will not proof. □

The last step or steps in the leaf set can often be avoided altogether. The routing algorithm assumes that we need to find the nearest node to a given ID. In most protocols data is not stored on a single node but on the k nearest nodes to that node to maintain reliable operation even in the case of failure of nodes.

2.2. **Insertion of Peers.** Suppose a new computer p wants to enter the network. Let R, L and M be its routing table, leaf set and neighbourhood set which will be filled using the following algorithm: p begins by generating its ID $id(p)$ and then contacting a peer p_0 already in the network. We assume that it is possible for p to do this and we assume that $d(p, p_0)$ is small. Now p_0 routes a *join*-message with recipient $id(p)$ along the standard routing mechanism of pastry. This message carries the address of p in the underlying network, e.g. its IP. Every peer that receives the message sends its routing table, leaf set and neighbourhood set to p. Let $p_0, p_1, ..., p_z$ be the path of the message. We assume that $z \geq m$ and $pfxl(p_i, p) \geq i$ for $i < m - 1$. If this is not the case we include peers multiple times.

$d(p, p_0)$ is small and therefore for all $q \in M_0 : d(p, q) \leq d(p, p_0) + d(p_0, q)$ is small. This means that M_0 is an appropriate choice for M. Because of the way how routing works in pastry $|id(p) - id(p_z)|$ is minimal. $L_z \cup \{p_z\}$ contains the peers with IDs numerically closest to $id(p)$. Therefore p can construct its leaf set using L_z and inserting p_z at position 1 or -1. Let R_i be the routing table of p_i. For p's routing table we copy the 0-th row from R_0, the first from R_1 and so on. This works because $pfxl(p_i, p) \geq i$ and therefore $\forall q \in R_i[i, *] : pfxl(q, p) \geq i \wedge pfxl(p, p_i) \geq i \implies pfxl(p, q) \geq i$. p now requests the routing tables of all peers in M and searches for entries that are better in terms of the metric d.

p sends messages about its arrival to all nodes in M, L and R. They check if p is better value for the appropriate entry in their routing table. The nodes in M may include p in their neighbourhood set. Nodes in L need to insert p into their leaf sets.

Notifying other nodes about the arrival of p costs $|M| + |L| + 2^b/b \log n$ messages. There were $\mathcal{O}(\frac{\log n}{b})$ replies to the *join*-message.

2.3. **Locality.** Normally (i.e. when using TCP/IP) to computers which are near in terms of the proximity metric are also physically near. This means that they are, depending on the size of the pastry-network, in the same building, city or country. This means that it is not unlikely that they will go down simultaneously. The reverse is true for nodes that are distant.

Because IDs are randomly chosen nodes in the leaf set of a node will be distributed randomly in the physical world. Therefore it is unlikely that all nodes on one side of the leaf set of a node will go down at the same time. But this is the only condition in which routing in pastry can fail.

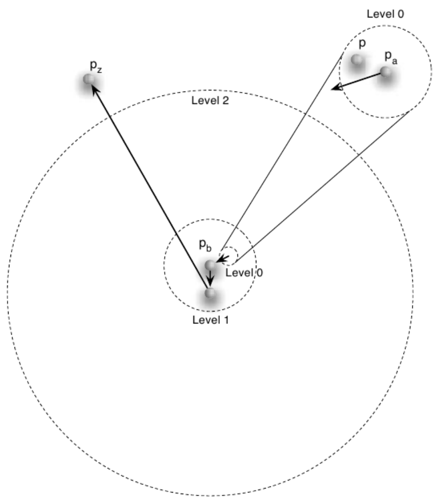
2.4. **Optimisation of Locality.** We will now show that routing in pastry is not only efficient in terms of the number of peers a message is routed through but also concerning the proximity metric of the underlying network. If there were no recent node failure routing is mostly done using the routing table. Also the constraints on the entries in the routing table are lighter than those for the leaf set. Therefore routing can be optimised by filling R_p for a node p only with "good" values, meaning that $d(q, p)$ is small for all $q \in R_p$.

We use the same names as two sections before. Assume that the routing tables of all nodes already in the network are optimised. $R[0, *]$ is taken from p_0. $d(p_0, q)$ is small for all $q \in R[0, *]$. With $d(p_0, p)$ being small we get that $d(p, q)$ is also small. $R[1, *]$ is taken from

p_1. Let q be now element of $R[1,*]$. $d(p_1, q)$ is optimal, but that does not imply that $d(p, q)$ is small. However $d(p, q)$ is relatively small because q must be in the set$\{s|pflx(s, p) \geq 1\}$. Because distances are distributed randomly through the ID-space here a lower higher value for $d(p, q)$ is still good. And so on for all $R[i,*]$.

The following picture illustrates this. (in the picture $a = 0$ and $b = 1$)

2.5. **Locality in Routing.** Let $p_1, ..., p_a, p_b, p_c, ..., p_n$be the path a certain message travels in pastry, so that p_b and p_c are chosen from the leaf sets of p_a and p_b respectively. We easily see that $d(p_a, p_b) < d(p_b, p_c)$ Otherwise ..., p_a, p_c , ... would be used as both p_b and p_c fullfill the requirements for the entry in the routing table of p_a where a reference to p_b is stored. Hence the distances increase monotonically. They even grow exponentially because every p_i is taken from a set with $n/2^{bi}$ entries. This means that any error we make on a low level will be far outweighed by the steps on the higher levels. As seen in the picture from the previous section we thereby get good results. Here one can see that with even if p and p_b are on opposite sites of the circle representing the leaf set of $p_a$the distance $d(p, p_b)$ is less than $\frac{1}{8}$ of what a message has to travel two steps later.



Note that the last steps are the most expensive ones. As the leaf set cannot be optimised for locality the final step (or steps) there will have a cost of about half the maximum of all distances. But this step can usually be avoided because we only need to reach one of the k nodes with IDs close to r.

Again this is not a formal proof and we will present experimental results to verify it.

3. Stability

3.1. **Leaf Set.** A peer may leave the network without warning leading to dead entries. The underlying network supports an operation that checks if a node is alive. A node is considered alive if the computer it is running on can still be contacted by all other peers and it is still running pastry. The protocol atop pastry may include keep alive messages. To reduce overhead they are only used when there was no recent normal message and only to maintain the leaf set.

If a peer p' fails to respond to pinging from another peer p, p' is assumed to be dead. p immediately tries to repair this. It requests the leaf sets from other nodes in its leaf set. With them p can fill its leaf set and reconstruct the leaf set of the now dead node. All nodes in $L_{p'}$ are notified by p and update their leaf sets appropriately.

3.2. **Routing Table.** The algorithm for insertion does not guarantee that all nodes are properly updated. Consider the insertion of a peer p. Only peers in $M_p \cup L_p \cup R_p$ are notified. Only nodes in L_p need to have a reference to p in their leaf sets, so this part is working as needed. However there may be peers that need to include p in their routing table but are not notified. As an example consider a network with no node with an ID with prefix 1. If $id(p)$ starts with 1 all nodes will need to be updated.

However correct routing is still guarantied. Also it is not probable that many nodes need to be updated. In the example the probability that there is no node with an ID with prefix 1 for a network with $|B| = 16$ and 15 nodes not including p is only .38 and here every node is in every other nodes leaf set, so all routing tables will need to be updated. As the number of nodes increases more nodes need to updated but at the same time the probability that there is no node with a certain prefix decreases.

If a peer notices that one of the entries in its routing table is *null* although there is an appropriate node in the network it searches for an appropriate value to fill the table. A peer may notice that a node fitting into a certain place in its routing table exists when a message to or from such a node is routed through p. If the protocol supports this p may also check the full path along which the message has travelled.

An alternate method can be seen in FreePastry. A peer p gets a message m routed from peer p'. Before routing m to the next peer p does the following:

```
Let $l \leftarrow pflx(id(p), id(m))$
if $pfxl(id(p'), id(m)) = l$ and
    $R[l, id(m)[l]] \neq null$ do
    Send our $R[l, *]$ to $p'$.
```

This second method has the advantage of performing better if there are a few routes which are mostly used. For example when a big file is transfered in multiple small parts the best route will be used for all but the first parts. The first method will be used in the remaining part of this paper.

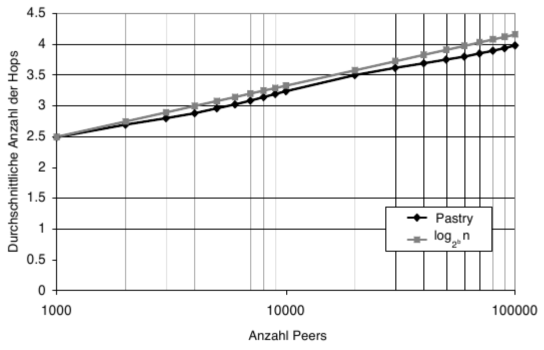
Before a peer p routes a message to a node p' it checks if p' is alive. If it is not routing continues as if the entry in p's routing table had been *null*. Then p tries to find a new entry to fill its routing table. To do this p asks all other $q \in R[i, *]$ for their entry $R_q[i, j]$. If this does not succeed it tries its next row $R[i + 1, *]$, $R[i + 2, *]$ and so on.

3.3. **Malicious Nodes.** So far we have only discussed reliability in cases where a node visibly fails. It may happen that a node still participates in the network but does not act accordingly to the protocols. It may for example accept messages but fail to forward them. This may be due to a faulty implementation of pastry or clients that try to interfere with the normal operation of the network. We assume that most nodes in the network are working properly.

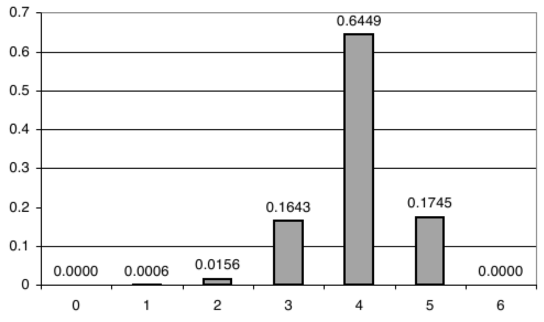
To avoid faulty nodes the first step is to be able to determine if messages reach their destination. To achieve this the protocol on top of pastry should include reply messages that are for example send after storing or modifying data. These messages should be signed. Because the node IDs are the hash values of the nodes public keys in a big network it is difficult to forge k replies from the k nodes that need to store a single piece of data.

If no reply is received after sending some request, the request is resend. In the normal implementation of pastry this new request will be routed along the same path as the original one and will therefore fail. To avoid this routing can be randomised: The algorithm for routing from a node p is only used in most cases, for example 95% of all routes. In the other cases a node q is chosen randomly. Because we still have to ensure that routing completes in finite time the following q has to be chosen in such a way that $pfxl(id(q), r) \geq pfxl(id(p), r)$ and$|id(q) - r| > |id(p) - r|$ are true.

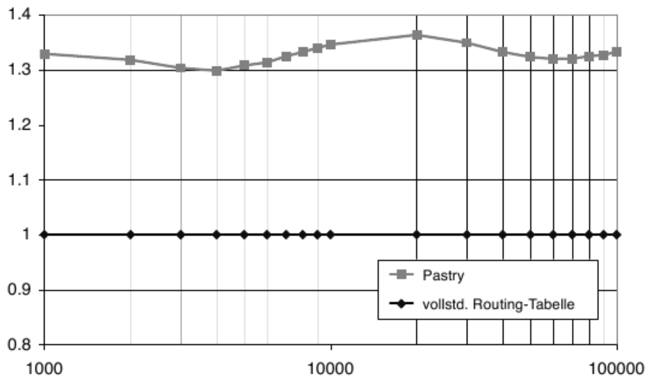
3.4. **Experimental Results.** We now present experimental results about the efficiency of pastry. This data was obtained with a simulation running on a single computer in one JVM. The nodes were uniformly distributed in $[0; 1000] \times [0; 1000]$ and distances were computed using the euclidian metric. Of course these are not realistic assumptions but they should provide results that will also apply to a certain degree to a network like the internet. In the experiments $b = 4$, so there are 16 digits, the size of the leaf set is 16 and the size of the neighbourhood set is set to 32. If not stated otherwise there are $100,000$ nodes in the network. In the first experiment we look at the number of hops needed for routing. This is almost logarithmic as expected.



The next figure shows the distribution of the number of hops. The maximum is $5 = \lceil \log_{2^b} 100,000 \rceil$.



We now look at the distances in the metric of the underlying network. For this experiment 200, 000 pairs of nodes are randomly selected and the distances are measured. They are then compared to the direct distance between the two nodes. Note that pastry scales very well in this respect.



4. Conclusion and Outlook

4.1. **PAST.** PAST is a distributed file storage system using pastry. As such it supports storing and retrieving files in a distributed and fail safe fashion. Changing files or deleting them is not supported. Files have IDs which are computed as their hash value. As said before a file is stored in the k nodes with nearest IDs to its own ID. Nodes that store a file have to make sure that a new copy is stored in the network when one of the k nodes goes down.

Also every node has a capacity that it contributes to the network. This capacity should not differ by a factor of more than 100. If a node that is about to join the network is too big, it joins the network as multiple nodes as if multiple instances of pastry were running on the computer. Usually the neighbourhood set and part of the routing table can be shared between these nodes. If on the other hand the node has a very low capacity it cannot join the network. Nodes may also join as observers in which case they can retrieve but not store data.

If a node n is among the k nodes that need to store a file but the node has already reached its capacity or is close to the file can be diverted. This means that n asks a node in its leaf set to store the file and then creates a link to this node. Nodes may store no more than one copy of any file. A request for storing a file fails if at least one node has reached its capacity and cannot divert the file. This can happen if the nodes in a part of the ID-space store relatively large amounts of data. In this case the file may be tried to be stored with a different ID.

Whenever a file is transfered through a node and that node has unused capacity the file is stored as cache. This may happen upon creation or retrieval of a file.

4.2. **Other Applications.** There are many other applications build on top of pastry. The list includes a publish/subscribe system (SCRIBE), a caching system (SQUIRREL), a messaging infrastructure (POST) and a a high-bandwidth content distribution system (SplitStream) that in turn uses SCRIBE.

In SCRIBE nodes can create topics which other nodes can subscribe to. Each topic has an ID that is computed as the hash value of its name. The node p with the ID nearest to that of a topic becomes the root of the topic. It stores the ID and the public key of the node which created the topic. k nodes with IDs nearest to that of p become back up roots for the topic. If p goes down one of them becomes the new root for the topic. When a node wants to subscribe to the topic it routes a *subscribe*-message to p. Let $q = p_m, p_{m-1}, ..., p_0$ be the path of that message. Each node p_i now stores a reference to p_{i+1}. We call such references children. If a *subscribe*-message is received by a node which already has children, it only adds the node the message came from as a child but does not forward the message. Using this procedure a tree with root p is generated. Whenever a new message arrives at p, usually send from the node that created the topic, p passes the message to its children. They in turn pass the message to their children and so on. Because routing takes into account locality, the same will be true for the multicast trees. Procedures are added to detect and repair failure of nodes in the tree.

In SplitStream this mechanism is generalised to provide better load balancing for large amounts of data such as streaming high definition videos. The data is split into multiple parts and each part is distributed among the nodes participating in the network using a multicast tree. Outbound bandwidth of the single nodes can also be taken into account.

References

[1] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. *SOSP'03*, October 2003.

[2] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. *NGC 2003*, September 2003.

[3] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. *HotOS VIII*, May 2001.

[4] Peter Mahlmann and Christian Schindelhauer. *Peer-to-Peer-Netzwerke*. Springer, 2007.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[6] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *18th ACM SOSP'01*, October 2001.

[7] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructur. *NGC2001*, November 2001.

, , , , , ,