PASTRY

Dheeraj Kumar Singh, Gautam Kumar, and Rasha Eqbal

Group 4

1 Introduction

Peer-to-peer networking is a distributed application architecture that partitions tasks or work loads between peers. Peers are equally privileged, equipotent participants in the application. There are no central servers involved, a reason for much of its popularity. One of the keys problems in large-scale peer-to-peer networks is providing efficient algorithms for object location and routing. Scalability is a major issue.

Pastry[1] is an object distribution and routing mechanism that is used for wide area peer-to-peer networks. It is scalable and decentralised and can automatically reorganise itself at the arrival, departure and failure of nodes. Its good route locality properties make it even more efficient. The algorithm used by Pastry tries to minimise the routing distance of a message. A message reaches its destination in number of steps which is logarithmic in the number of Pastry nodes present. All these factors make Pastry a good choice for peer-to-peer networks.

2 Design

Each node in the Pastry peer-to-peer network is assigned a 128 bit node ID. This node ID is assigned randomly and could be computed as the cryptographic hash of the node's public key or its IP address. The IDs are generated in such a way that they are uniformly distributed in the node ID space. As a result of this kind of distribution nodes with adjacent node IDs are widely spread out with a high probability.

Given a message and a key, the routing takes place as follows. The node IDs and the key can be thought of as a sequence of digits in the base 2^b system, where b is a network configuration parameter, typically chosen as 2 or 3. When a node receives the message, it forwards it to a node whose ID shares a prefix longer in length than the prefix the key shares with the current node ID. If such a node is not found, the message is forwarded to a node whose shared prefix length is equal to the current shared prefix length, but which is numerically closer to the key. As such the message is delivered to the node whose ID is closest to the key in less than $\lceil \log_{2^b} N \rceil$ steps where N is the total number of Pastry nodes. The system is pretty robust guaranteeing delivery even in the case of failure of adjacent nodes. The details of implementation are explained in the next sections.

3 Node State

Each Pastry node maintains a routing table R, a neighbourhood set M and a leaf set L.

The Routing Table R: The routing table of a node consists of $\lceil \log_{2^b} N \rceil$ rows and $2^b - 1$ columns. The row n contains entries for nodes whose IDs share the first n digits with the current node ID but are different in the $n+1^{th}$ digit. The $2^b - 1$ columns correspond to these $2^b - 1$ different values of the $n + 1^{th}$ digit except the value of the $n + 1^{th}$ digit of the current node ID. The table entry contains the IP address of such a node. If there are more than one node with this property the closest one is chosen based on the proximity metric. If no such node exists the entry is left blank.

The Neighbourhood Set M: This contains a list of node IDs and their corresponding IP addresses. The nodes are closest to the current node based on the proximity metric. Though this is not used for the routing purposes, it is used to maintain the locality properties.

The Leaf Set L: This contains a list of nodes with numerically closest node IDs with respect to the current node, $|L|/2$ nodes with node IDs smaller than the current node and $|L|/2$ nodes with node IDs larger than the current node. This list is used for routing. Typical values for $|M|$ and $|L|$ are 2^b or 2×2^b.

4 Routing

Whenever a message with a key D arrives at a node the steps followed by the node are as follows.

The node first checks if D falls in the list of node IDs in its leaf set L. If yes it directly forwards the message to the required node.

If such a node is not present in the leaf set, the routing table is used to find a node whose node ID shares a common prefix with the key longer by at least one more digit than the prefix shared by the current node ID. The message is then forwarded to this node.

If such a node does not exist, i.e. the corresponding entry in the routing table is empty or there is a node failure, the message is forwarded to a node whose ID shares a common prefix with the key at least as long as the current node ID but is numerically closer to the key than the current node ID. Such a node will exist in the leaf set unless the message has already arrived at a node with the numerically closest node ID. And one of these nodes will be alive unless $|L|/2$ nodes fail simultaneously. So such a node can always be found.

The routing algorithm will converge because at each step we are choosing one of two types of nodes:

1. Node whose node ID shares a common prefix with the key at least one digit longer than the common prefix shared by the current node ID.
2. Node whose node ID shares a common prefix with the key at least as long as the current node ID but numerically closer to the key.

The pseudocode for the routing algorithm is given below. The notations used are:

- A: the current node ID.
- R_l^i: entry in the routing table R at column i of row l.
- L_i: i^{th} closest node ID in the leaf set L. Negative indices indicate smaller node IDs and positive indices indicate larger node IDs.
- D_l: the l^{th} digit in the key D.
- $shl(A, B)$: length of the prefix shared among A and B.

Pseudocode for Routing:

if $(L_{-|L|/2} \leq D \leq L_{|L|/2})$ **then**
  //D is in the range of the leaf set
  Forward to L_i such that $|D - L_i|$ is minimal
else
  // search for a node with longer shared prefix in the routing table
  Let $l = shl(A, D)$
  if $(R_l^{D_l} \neq null)$ **then**
    forward to $R_l^{D_l}$
  else
    // empty entry or failed node in the routing table
    forward to $T \in L \cup R \cup M$, s.th.
      $shl(T, D) \geq l$,
      $|T - D| < |A - D|$
  end if
end if

4.1 Performance Analysis

Assuming accurate routing tables and no recent node failures, the expected number of routing steps given N Pastry nodes is $\lceil \log_{2^b} N \rceil$. Consider three different cases:

- When the routing table is used to find a longer common prefix, we reduce the number of possible nodes by a factor of 2^b. Hence the destination can be reached in $\lceil \log_{2^b} N \rceil$ steps.
- When we have the key within the ID range of the nodes in the leaf set the destination is just one step away.
- When neither of these cases occurs, a node with the required prefix does not exist. The likelihood of this happening given the uniform distribution of IDs depends on $|L|$. Analysis shows that with $|L| = 2^b$ and $|L| = 2 \times 2^b$, the probability that this case arises during a given message transmission is less than .02 and 0.006, respectively. When it happens, no more than one additional routing step results with high probability.

In the event of many simultaneous node failures, the number of routing steps required may be linear in N in the worst case. However in practice, routing

performance degrades gradually with recent node failures and message delivery is assured given $|L|/2$ consecutive node IDs do not fail simultaneously. With a suitable choice of $|L|$ and the expected diversity in node IDs resulting from a uniform distribution in the node ID space, this failure probability can be made very low.

5 Self - Organization

5.1 Node Arrival

When a new node arrives it has to initialize its state and inform other nodes of its presence. We assume that we already have information about a nearby existing node A according to the proximity metric. Such a node can be automatically located by expanding ring IP multicast, or can be obtained by the system administrator through outside channels.

The node ID of the newly joining node X is computed usually as the $SHA-1$ hash of its public key or IP address. Node X asks A to route a message "join" with the key X. As is expected, this message is then routed to the node Z whose ID is numerically closest to X's. In response to this message routing, nodes A and Z and all other nodes encountered on the path send their state information to X. X can then use this information along with state information requested from additional nodes to initialize its state. Finally it informs all the nodes that need to be aware of its presence, so that they can update their state tables too.

A is assumed to be in the proximity of X, so X's neighbourhood list can be initialized using A's. Node Z has node ID numerically closest to X, so Z's leaf set can be used for the leaf set initialization of X. Now only the routing table of X needs to be initialized. Let us consider the most general case when the node IDs of A and X share no common prefix. The zero-th row of X's routing table can be the same as that of A's. Since the zero-th row contains entries for nodes with no common prefix, it is independent of the node ID. So we can safely copy A's zero-th row as X's zero-th row. The next node encountered on the path to Z will have an ID sharing a prefix of at least length one with the key. So the first row of this node's routing table can be the first row of X's routing table. Similarly, the next node in the path will have an ID having common prefix of length two, so X's routing table can have as its second row the second row of this node's routing table. This process is carried on until the routing table of X is fully created.

After this, X transmits its state information to all the nodes in its routing table, leaf set and neighbourhood set, so that they can update their states accordingly. The total cost for this join, in terms of the messages exchanged, is $O(\log_{2^b} N)$ with the constant being 3×2^b.

An optimistic approach is used for concurrent node joins. This is appropriate as the arrival of a node affects only a small number of nodes in the network. When an existing node B sends its state information to a new node A, it attaches a timestamp to it. Node A then initialises its state accordingly and then sends

its updated state back to B with a timestamp. It also appends the previously received timestamp. Node B then checks if its state has been updated after the time indicated by the old timestamp received. If yes, it resends its state information to A. A then restarts its operation accordingly.

5.2 Node Departure or Failure

A node is said to have failed when its immediate neighbours in the node ID space are no longer able to communicate with it.

To replace a failed node in the leaf set say L_i, its neighbour contacts the node with largest index on the side of the failed node to get its leaf set. Say $-|L|/2 \leq i \leq 0$, then the leaf set of $L_{-|L|/2}$ is obtained. This set will contain nodes overlapping with the current leaf set and some new nodes. A node is chosen from among the new ones to replace the failed node. First it is required to check whether the node we are replacing the failed node with is still alive by contacting it. The leaf sets are thus lazily repaired unless $|L|/2$ nodes with adjacent IDs have failed simultaneously. Given the uniform random distribution of the IDs and the diversity of nodes with adjacent node IDs, the probability of this happening is very low.

A failure in the routing table occurs when a node tries to contact another one for forwarding the message and gets no response. We have seen that this causes no delay in routing; another node is chosen to forward the message. However we need to update the routing table to maintain its integrity. Suppose a node corresponding to entry R_l^d has failed. We contact another node with an entry R_l^i in the same row, and ask for the R_l^d entry in this node's routing table to replace the failed node. If no such node is the same row is alive, we move to the next row and find an R_{l+1}^i. We continue this until we get a node entry to replace the failed node. If such a node exists we will most likely be able to locate it using this procedure.

A failed node in the neighbourhood set does not directly affect routing, but we need to replace it anyway as it plays a role in exchanging locality information. Hence a node periodically checks if the nodes in its neighbourhood set are alive. If a node has failed, it contacts its other neighbours and uses the newly discovered nodes in their neighbourhood sets, to find a node least distance away. It uses this to replace the failed node.

5.3 Locality

It is essential, that the route that is chosen is likely to be "good" using a given proximity metric. Given this property holds before, it should be ensured that it holds after the arrival of a new node. A new node X asks an existing node A to route a message with X as the key. Say, the route is $A, B, ..., Z$. Node X's routing table's i^{th} row is populated by the i^{th} node encountered on this route.

Since entries in row 0 of A's table are close to A, and A is close to X therefore, entries are relatively near A. The same goes for other entries in the table. However, it is essential that the relative closeness is improved to avoid cascading

errors. For this, there is a second phase in which X asks state from the nodes in its routing table and neighbourhood set. It compares the distance of corresponding entries in nodes' routing table and neighbourhood sets and updates its entries with closer nodes. In this way, however the route selected might not be the shortest route, it selects relatively good routes. Also, the expected distance that is traveled by a messages during successive steps is increasing exponentially since the difference occurs at a more significant digit and due to the distribution of nodeIds in the network, nodes that differ at more significant digits are more distant.

6 Conclusion

Peer-to-peer networks have gained popularity over the recent years. Gnutella, Freenet and Napster are some traditional peer-to-peer systems that have provided the motivation for further development. In recent years, CAN, Chord, Pastry and Tapestry have evolved as decentralised solutions to the problem. In these solutions there is an inherent trade-off between the information a node has to maintain (essentially dictating how centralised the system is) and the routing time. Pastry is an efficient scheme with both routing time and space requirements of the order of $O(log_{2^b} N)$.

References

1. Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.