Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems

Antony Rowstron[1] and Peter Druschel[2]*

[1] Microsoft Research Ltd, St. George House,
Guildhall Street, Cambridge, CB2 3NH, UK.
antr@microsoft.com
[2] Rice University MS-132, 6100 Main Street,
Houston, TX 77005-1892, USA.
druschel@cs.rice.edu

Abstract. This paper presents the design and evaluation of Pastry, a scalable, distributed object location and routing substrate for wide-area peer-to-peer applications. Pastry performs application-level routing and object location in a potentially very large overlay network of nodes connected via the Internet. It can be used to support a variety of peer-to-peer applications, including global data storage, data sharing, group communication and naming.

Each node in the Pastry network has a unique identifier (nodeId). When presented with a message and a key, a Pastry node efficiently routes the message to the node with a nodeId that is numerically closest to the key, among all currently live Pastry nodes. Each Pastry node keeps track of its immediate neighbors in the nodeId space, and notifies applications of new node arrivals, node failures and recoveries. Pastry takes into account network locality; it seeks to minimize the distance messages travel, according to a to scalar proximity metric like the number of IP routing hops.

Pastry is completely decentralized, scalable, and self-organizing; it automatically adapts to the arrival, departure and failure of nodes. Experimental results obtained with a prototype implementation on an emulated network of up to 100,000 nodes confirm Pastry's scalability and efficiency, its ability to self-organize and adapt to node failures, and its good network locality properties.

1 Introduction

Peer-to-peer Internet applications have recently been popularized through file sharing applications like Napster, Gnutella and FreeNet [1,2,8]. While much of the attention has been focused on the copyright issues raised by these particular applications, peer-to-peer systems have many interesting technical aspects like decentralized control, self-organization, adaptation and scalability. Peer-to-peer systems can be characterized as distributed systems in which all nodes have identical capabilities and responsibilities and all communication is symmetric.

There are currently many projects aimed at constructing peer-to-peer applications and understanding more of the issues and requirements of such applications and systems [1,2,5,8,10,15]. One of the key problems in large-scale peer-to-peer applications

* Work done in part while visiting Microsoft Research, Cambridge, UK.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

is to provide efficient algorithms for object location and routing within the network. This paper presents Pastry, a generic peer-to-peer object location and routing scheme, based on a self-organizing overlay network of nodes connected to the Internet. Pastry is completely decentralized, fault-resilient, scalable, and reliable. Moreover, Pastry has good route locality properties.

Pastry is intended as general substrate for the construction of a variety of peer-to-peer Internet applications like global file sharing, file storage, group communication and naming systems. Several application have been built on top of Pastry to date, including a global, persistent storage utility called PAST [11,21] and a scalable publish/subscribe system called SCRIBE [22]. Other applications are under development.

Pastry provides the following capability. Each node in the Pastry network has a unique numeric identifier (nodeId). When presented with a message and a numeric key, a Pastry node efficiently routes the message to the node with a nodeId that is numerically closest to the key, among all currently live Pastry nodes. The expected number of routing steps is O(log N), where N is the number of Pastry nodes in the network. At each Pastry node along the route that a message takes, the application is notified and may perform application-specific computations related to the message.

Pastry takes into account network locality; it seeks to minimize the distance messages travel, according to a scalar proximity metric like the number of IP routing hops. Each Pastry node keeps track of its immediate neighbors in the nodeId space, and notifies applications of new node arrivals, node failures and recoveries. Because nodeIds are randomly assigned, with high probability, the set of nodes with adjacent nodeId is diverse in geography, ownership, jurisdiction, etc. Applications can leverage this, as Pastry can route to one of k nodes that are numerically closest to the key. A heuristic ensures that among a set of nodes with the k closest nodeIds to the key, the message is likely to first reach a node "near" the node from which the message originates, in terms of the proximity metric.

Applications use these capabilities in different ways. PAST, for instance, uses a fileId, computed as the hash of the file's name and owner, as a Pastry key for a file. Replicas of the file are stored on the k Pastry nodes with nodeIds numerically closest to the fileId. A file can be looked up by sending a message via Pastry, using the fileId as the key. By definition, the lookup is guaranteed to reach a node that stores the file as long as one of the k nodes is live. Moreover, it follows that the message is likely to first reach a node near the client, among the k nodes; that node delivers the file and consumes the message. Pastry's notification mechanisms allow PAST to maintain replicas of a file on the k nodes closest to the key, despite node failure and node arrivals, and using only local coordination among nodes with adjacent nodeIds. Details on PAST's use of Pastry can be found in [11,21].

As another sample application, in the SCRIBE publish/subscribe System, a list of subscribers is stored on the node with nodeId numerically closest to the topicId of a topic, where the topicId is a hash of the topic name. That node forms a rendez-vous point for publishers and subscribers. Subscribers send a message via Pastry using the topicId as the key; the registration is recorded at each node along the path. A publisher sends data to the rendez-vous point via Pastry, again using the topicId as the key. The rendez-vous point forwards the data along the multicast tree formed by the reverse paths

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

from the rendez-vous point to all subscribers. Full details of Scribe's use of Pastry can be found in [22].

These and other applications currently under development were all built with little effort on top of the basic capability provided by Pastry. The rest of this paper is organized as follows. Section 2 presents the design of Pastry, including a description of the API. Experimental results with a prototype implementation of Pastry are presented in Section 3. Related work is discussed in Section 4 and Section 5 concludes.

2 Design of Pastry

A Pastry system is a self-organizing overlay network of nodes, where each node routes client requests and interacts with local instances of one or more applications. Any computer that is connected to the Internet and runs the Pastry node software can act as a Pastry node, subject only to application-specific security policies.

Each node in the Pastry peer-to-peer overlay network is assigned a 128-bit node identifier (nodeId). The nodeId is used to indicate a node's position in a circular nodeId space, which ranges from 0 to $2^{128} - 1$. The nodeId is assigned randomly when a node joins the system. It is assumed that nodeIds are generated such that the resulting set of nodeIds is uniformly distributed in the 128-bit nodeId space. For instance, nodeIds could be generated by computing a cryptographic hash of the node's public key or its IP address. As a result of this random assignment of nodeIds, with high probability, nodes with adjacent nodeIds are diverse in geography, ownership, jurisdiction, network attachment, etc.

Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\lceil log_{2^b} N \rceil$ steps under normal operation (b is a configuration parameter with typical value 4). Despite concurrent node failures, eventual delivery is guaranteed unless $\lfloor |L|/2 \rfloor$ nodes with *adjacent* nodeIds fail simultaneously ($|L|$ is a configuration parameter with a typical value of 16 or 32). In the following, we present the Pastry scheme.

For the purpose of routing, nodeIds and keys are thought of as a sequence of digits with base 2^b. Pastry routes messages to the node whose nodeId is numerically closest to the given key. This is accomplished as follows. In each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node's id. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's id. To support this routing procedure, each node maintains some routing state, which we describe next.

2.1 Pastry Node State

Each Pastry node maintains a *routing table*, a *neighborhood set* and a *leaf set*. We begin with a description of the routing table. A node's routing table, R, is organized into $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries each. The $2^b - 1$ entries at row n of the routing table each refer to a node whose nodeId shares the present node's nodeId in the first n digits,

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

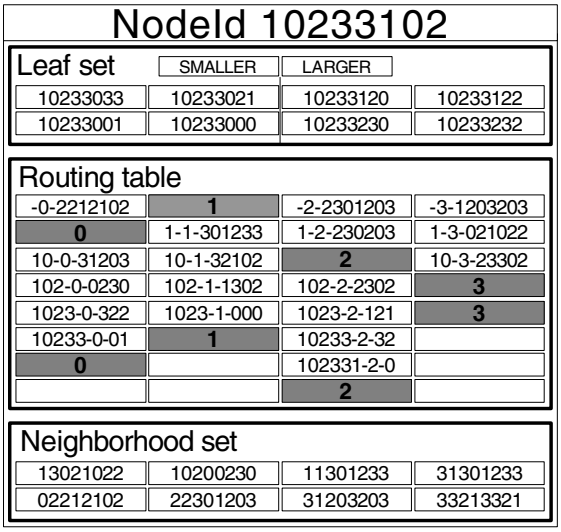

Fig. 1. State of a hypothetical Pastry node with nodeId 10233102, $b = 2$, and $l = 8$. All numbers are in base 4. The top row of the routing table is row zero. The shaded cell in each row of the routing table shows the corresponding digit of the present node's nodeId. The nodeIds in each entry have been split to show the *common prefix with 10233102 - next digit - rest of nodeId*. The associated IP addresses are not shown.

but whose $n+1$th digit has one of the $2^b - 1$ possible values other than the $n+1$th digit in the present node's id.

Each entry in the routing table contains the IP address of one of potentially many nodes whose nodeId have the appropriate prefix; in practice, a node is chosen that is close to the present node, according to the proximity metric. We will show in Section 2.5 that this choice provides good locality properties. If no node is known with a suitable nodeId, then the routing table entry is left empty. The uniform distribution of nodeIds ensures an even population of the nodeId space; thus, on average, only $\lceil log_{2^b} N \rceil$ rows are populated in the routing table.

The choice of b involves a trade-off between the size of the populated portion of the routing table (approximately $\lceil log_{2^b} N \rceil \times (2^b - 1)$ entries) and the maximum number of hops required to route between any pair of nodes ($\lceil log_{2^b} N \rceil$). With a value of $b = 4$ and 10^6 nodes, a routing table contains on average 75 entries and the expected number of routing hops is 5, whilst with 10^9 nodes, the routing table contains on average 105 entries, and the expected number of routing hops in 7.

The neighborhood set M contains the nodeIds and IP addresses of the $|M|$ nodes that are closest (according the proximity metric) to the local node. The neighborhood set is not normally used in routing messages; it is useful in maintaining locality properties, as discussed in Section 2.5. The leaf set L is the set of nodes with the $|L|/2$ numerically closest larger nodeIds, and the $|L|/2$ nodes with numerically closest smaller nodeIds, relative to the present node's nodeId. The leaf set is used during the message routing, as described below. Typical values for $|L|$ and $|M|$ are 2^b or 2×2^b.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

How the various tables of a Pastry node are initialized and maintained is the subject of Section 2.4. Figure 1 depicts the state of a hypothetical Pastry node with the nodeId 10233102 (base 4), in a system that uses 16 bit nodeIds and a value of $b = 2$.

2.2 Routing

The Pastry routing procedure is shown in pseudo code form in Table 1. The procedure is executed whenever a message with key D arrives at a node with nodeId A. We begin by defining some notation.
R_l^i: the entry in the routing table R at column $i, 0 \leq i < 2^b$ and row $l, 0 \leq l < \lfloor 128/b \rfloor$.
L_i: the i-th closest nodeId in the leaf set L, $-\lfloor |L|/2 \rfloor \leq i \leq \lfloor |L|/2 \rfloor$, where negative/positive indices indicate nodeIds smaller/larger than the present nodeId, respectively.
D_l: the value of the l's digit in the key D.
$shl(A, B)$: the length of the prefix shared among A and B, in digits.

Table 1. Pseudo code for Pastry core routing algorithm.

```
(1)  if (L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋}) {
(2)      // D is within range of our leaf set
(3)      forward to L_i, s.th. |D − L_i| is minimal;
(4)  } else {
(5)      // use the routing table
(6)      Let l = shl(D, A);
(7)      if (R_l^{D_l} ≠ null) {
(8)          forward to R_l^{D_l};
(9)      }
(10)     else {
(11)         // rare case
(12)         forward to T ∈ L ∪ R ∪ M, s.th.
(13)             shl(T, D) ≥ l,
(14)             |T − D| < |A − D|
(15)     }
(16) }
```

Given a message, the node first checks to see if the key falls within the range of nodeIds covered by its leaf set (line 1). If so, the message is forwarded directly to the destination node, namely the node in the leaf set whose nodeId is closest to the key (possibly the present node) (line 3).

If the key is not covered by the leaf set, then the routing table is used and the message is forwarded to a node that shares a common prefix with the key by at least one more digit (lines 6–8). In certain cases, it is possible that the appropriate entry in the routing table is empty or the associated node is not reachable (line 11–14), in which case the message is forwarded to a node that shares a prefix with the key at least as long as the local node,

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.