PAST: Persistent and Anonymous Storage in a Peer-to-Peer Networking Environment

Peter Druschel[1] and Antony Rowstron[2]
[1]Rice University, 6100 Main Street, MS 132, Houston, TX 77005-1892, USA*
[2]Microsoft Research Ltd., St. George House, 1 Guildhall Street, Cambridge, CB2 3NH, UK.

Abstract

This paper describes PAST, a large-scale Internet based storage utility that provides high availability, persistence and privacy of stored content, and protects the anonymity of clients and storage providers. PAST is a peer-to-peer Internet application and is entirely self-organizing. PAST nodes contribute storage to the system, serve as access points for clients, and participate in the routing of client requests. Nodes are not trusted, they may join the system at any time and may silently leave the system without warning. Yet, the system is able to provide strong assurances, efficient storage access, and scalability.

One of the most interesting aspects in the design of PAST is the use of smart cards, which are issued by a third party called a *broker*. The smart cards form the system's trusted computing base (TCB), and play a key role in PAST's ability to provide strong assurances, scalability, and efficiency, despite the weak assumptions about the behavior and trustworthiness of individual nodes. In this position paper, we will argue that the use of a broker substantially simplifies the construction of a robust and secure peer-to-peer storage service like PAST.

1 Introduction

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, 3, 4]. Peer-to-peer systems can be characterized as distributed systems in which all nodes have identical capabilities and responsibilities and all communication is symmetric.

We are currently developing PAST, an Internet based, peer-to-peer global storage utility, which aims to provide strong persistence, high availability, scalability, content privacy and anonymity of clients and storage providers.

While PAST offers persistent storage services, its access semantics differ from that of a conventional filesystem. Documents stored in PAST are associated with a *docId* that is quasi-uniquely associated with the document's content[1]. This implies that documents stored in PAST are *immutable* since a modified version of a document cannot be written with the same docId as its original.

PAST does not support a *delete* operation for documents. Instead, the owner of a document may *reclaim* the storage associated with a document. While the semantics of document deletion indicate that the document is removed when the operation completes, reclaim has weaker semantics, and simply means that a user can reuse the space, and the system no longer provides any guarantees about the availability of the document.

The PAST system is composed of nodes, where each node is capable of storing documents and routing client requests to insert, retrieve, or reclaim a document. The nodes form a self-organizing network. Inserted documents are replicated across multiple nodes. The system ensures, with high probability, that the set of nodes over which a document is replicated is diverse in terms of geographic location, ownership, administrative entity, network connectivity, rule of law and so forth.

An efficient routing protocol ensures that client requests to *insert* or *reclaim* a document are routed to each node that stores the document. Client requests to *retrieve* a document are routed to a node that is a good approximation to the"closest in the network"[2] to the client that issued the request, among all live nodes that store the requested document. The number of PAST nodes traversed, as well as the number of messages exchanged while routing a client request is at most logarithmic in the total number of PAST nodes in the system.

Whilst many aspects of PAST are interesting, one of the most fundamental design decisions is the adoption of a third party (broker) that issues smart cards for all users of the system. These smart cards provide a TCB, and allow PAST to provide strong assurances (e.g., persistence, privacy, anonymity) with high efficiency, despite weak assumptions about the behaviour and trustworthiness of PAST nodes and limited trust placed in the broker.

The broker is not involved in the operation of the PAST network and knows nothing other than a mapping of operators to smart card ids, how much storage a client is able to use and how much storage a node operator has

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

[1]The docId is based on a secure hash of the document's content. Therefore, it is extremely unlikely that documents with different contents have the same docId.

[2]The notion of network proximity may be based on geographic location, number of network hops, bandwidth, delay, or a combination of these and other factors.

agreed to contribute to the system.

One of the main issues of peer-to-peer systems, and particularly storage or document-sharing systems, is privacy and anonymity. A provider of storage space used by others does not want to risk prosecution for content of the documents it stores, and a client inserting or retrieving a document may not wish their identity to be known by others. Anderson [5] describes the "the Gutenberg Inheritance" and motivates why such levels of privacy and anonymity are required.

PAST clients and storage providers need not trust each other, but place limited trust in the broker. In particular, clients and storage providers trust that the broker facilitates the operation of a functioning PAST network. However, clients need not reveal to the broker (or anyone else) the content they are storing or anything about their usage of the system, except for the total amount of storage available to them. Similarly, neither the broker nor anyone other than the node itself knows which docIds are stored at a storage node.

In the following sections, we shall present enough detail about the design of PAST to make the case that the use of a broker that issues smart cards facilitates the operation of an efficient, robust, and secure system that meets these requirements.

2 PAST architecture

Some of the key aspects of PAST's architecture are (1) the broker, which facilitates the secure operation of the system, (2) the use of smart cards, which form the system's TCB, (3) the heavy use of randomization to provide strong (probabilistic) assurances without the need for centralized control or expensive distributed agreement protocols, and (4) an efficient routing protocol that routes client requests in $O(logN)$ steps in a self-configuring topology.

The purpose of the broker is to certify (and protect) the identity of clients and storage node operators, to ensure a balance of demand and supply of storage space, and to issue smart cards, which form the system's TCB. The broker facilitates, but is not involved in, the interaction between the clients and the storage providers, much in the way a certification authority facilitates the authentication and secure communication between a Web browser and a secure (SSL) Web site. Clients pay the broker for the right to use storage space, whilst the broker pays the storage providers for space.

PAST is composed of nodes, where, in general, each node can be considered as both a storage node and a client. The smart card issued to the node is initially used to create a node identifier (nodeId), which is a 128 bit number chosen randomly from a uniform distribution. An expiration date is associated with the new nodeId, and this information is signed by the smart card.

Documents that are inserted into the PAST system are each assigned a docId. A docId is 160 bits in length, and is the secure hash (SHA-1) of a textual document name, a secure hash (SHA-1) of the content, and the creator's smartcard id. Before a document is inserted, a write certificate is generated, which contains the docId, document expiry date, the replication factor, the creation date and a secure hash of the content. The write certificate is signed by the smartcard of the document's creator.

When a documented is inserted in PAST, the network routes the document to the k nodes whose node identifiers are numerically closest to the first 128 bits of the document identifier (docId). Each of these nodes then stores a copy of the document. The replication factor k depends on the availability and persistence requirements of the document and may vary between documents.

This simple insertion procedure ensures that (1) the document remains available as long as one of the k nodes that store the document is alive; that (2) the set of nodes that store the document are diverse in geographic location, administration, ownership, network connectivity, rule of law, etc.; and, (3) that the storage requirements are balanced among storage nodes. (1) follows from the properties of the PAST routing algorithm described in Section 3. (2) and (3) follow from the independent drawing of a uniformly distributed random nodeId by each storage site and the properties of a secure hash function (uniform distribution of hash values, regardless of the set of documents).

2.1 Security and the role of the broker and smartcards

In discussing PAST's security, we make the following assumptions. We assume that it is computationally infeasible for an attacker to break the public-key cryptosystem and the secure hash function used by the smartcards. It is assumed that an attacker can control individual PAST nodes, but that they cannot control the behavior of the smartcard. Finally, it is assumed that the broker does not conspire with clients or storage node operators to violate the system's security.

The broker is a key concept in PAST's security model. It allows PAST to provide strong assurances about persistence, availability, anonymity, and privacy despite the fact that PAST nodes are not trusted. The broker acts as entity that is trusted (to a certain degree) by all parties, and thus serves as a certification authority. Moreover, the broker issues smartcards, which collectively form PAST's TCB.

The smartcard is issued by a PAST broker, and is used to generate unique ids for nodes, generate document capabilities, generate write/reclaim certificates and maintain client storage quotas. In the following, we discuss each of these functions.

Secure generation of nodeIds The smartcard generates and signs a nodeId when a node first joins the system. A secure random number generator embedded in the smartcard is used for this purpose. This ensures uniform coverage of the space of nodeIds, and a random spread of nodeIds across geographic locations, countries, node operators, etc. Furthermore, the use of signed

nodeIds prevents attacks involving malicious node operators trying to choose particular values for their nodeIds (for instance, to control all nodes that store a particular document).

Secure generation of write certificates and receipts The smartcard of a client wishing to insert a document into PAST issues a write certificate. The certificate contains a secure hash of the document's contents (computed by the client node, not the smartcard), the docId (computed by the smartcard), a replication factor, a document expiration date, and is signed by the smartcard. During an insert operation, the write certificate allows each storing node to verify that (1) the client is authorized to insert the document into the system, (2) the contents of the document arriving at the storing node have not been corrupted en route from the client, and (3) the docId is valid (it is consistent with the content arriving at the node). Each storage node that has successfully stored a copy of the document then issues a write confirmation that is returned to the client, which allows the client to (4) verify that k copies of the document have been created. (1) prevents clients from exceeding their storage quotas, (2) renders ineffective attacks based on malicious nodes involved in the routing of an insert request that change the content, (3) prevents denial-of-service attacks where malicious clients try to exhaust a subset of PAST storage nodes by generating bogus docIds with nearby values, and (4) prevents a malicious node from suppressing the creation of k diverse replicas. During a retrieve operation, the write certificate is returned along with the document, and allows the client to verify that the content has not been corrupted.

Secure generation of reclaim certificates and receipts Prior to issuing a reclaim operation, the client's smartcard generates a reclaim certificate. The certificate contains the docId, the client's smartcard id, and is signed by the smartcard. The certificate is included with the reclaim request, which is routed to the nodes that store the document. Upon processing a reclaim operation by a client, the smartcard of a storage node first verifies that the smartcard id in the reclaim certificate matches that in the write certificate stored with the document. This prevents clients other than the owner of the document from reclaiming the document's storage. If the reclaim operation is accepted, the smartcard of the storage node generates a reclaim certificate. The certificate contains the amount of storage reclaimed, the owner's smartcard id, and is signed by the smartcard and returned to the client.

Client storage quotas The smartcard maintains storage quotas. Each client smartcard is issued by the broker with an initial quota, depending on how much storage the client has purchased. When a write certificate is issued, an amount corresponding to the document size times the replication factor is debited against the quota. When the client presents an appropriate reclaim certificate issued by a storage node, the amount reclaimed is credited against the client's quota. This prevents clients from exceeding the storage quota they have paid for.

In the following, we discuss the role of the broker and smartcards in ensuring some of the system's key properties.

Providing system integrity Several conditions underly the basic integrity of a PAST system. Firstly, to maintain proper load balancing among storage nodes, the nodeIds and docIds must be uniformly distributed. The smartcards ensure that malicious nodes and clients cannot bias this distribution. Secondly, there must be a balance between the sum of all client quotas (potential demand) and the total available storage in the system (supply). The broker ensures that balance, using the price of storage to regulate supply and demand. Thirdly, individual malicious nodes must be incapable of persistently denying service to a client. A randomized routing protocol, described in Section 3, ensures that a retried operation will eventually by routed around the malicious node.

Providing Persistence Document persistence in PAST depends primarily on three conditions. (1) Unauthorized clients are prevented from reclaiming a document's storage, (2) the document is initially stored on k storage nodes, and (3) there is sufficient diversity in the set of storage nodes that store a document. By issuing and requiring reclaim certificates, the smartcards ensure condition (1). (2) is enforced through the use of write confirmations and (3) is ensured due to the random distribution of nodeIds, which can't be biased by an attacker.

Providing data privacy and integrity Clients use encryption to protect the privacy of their data. Encryption is performed with a cryptosystem of the client's choice, and does not involved the smartcards. Data integrity is ensured by means of the write certificates issued by the smartcards.

Providing anonymity Anonymity of clients in PAST is ensured because a client's smartcard id is the only information associating a stored document or a request with the responsible client. The association between a smart card id and the client's identity is only known to the broker. Anonymity of storage nodes is similarly guaranteed because only the broker knows the association between the node's smartcard id and the identity of the node operator. In addition, a small number of "neighboring" nodes know the IP address of a PAST node. However, when a PAST node forwards a request from or to a neighboring node, it has no way to determine if that node is the originator or destination of the request[3]. Therefore, it cannot be sure if the neighbor in question acts as client or storage node with respect to the request.

While space limitations prevent us from a full discussion of PAST's security model, we believe that our

[3]Except in cases where the docId matches the nodeId of the neighbor perfectly. This case is so unlikely that we ignore it.

overview shows that the use of a broker and the associated smartcards allow PAST to provide strong assurances that would be difficult to attain otherwise. Without a broker and a TCB, one would have to rely on voting protocols that are costly (in terms of the number of messages exchanged) and complex.

It is to be noted that multiple PAST systems, with separate brokers, can co-exist in the Internet. In fact, we envision multiple competing brokers, where a client can access documents in the entire system, but can only store documents on storage nodes affiliated with the client's broker. Furthermore, it is possible to operate isolated PAST systems that serve a mutually trusting community without a broker. In these cases, a virtual private network (VPN) can be used to interconnect the system's nodes.

In the remainder of this paper, we give a brief overview of other interesting aspects of PAST, namely its routing and self-configuration algorithms.

3 Routing

We now describe the routing scheme used by the PAST nodes to forward request messages. The nodeId is used to indicate a node's position in the namespace, which ranges from 0 to $2^{128} - 1$. A nodeId is sub-divided into a sequence of levels, where each level is represented by b contiguous bits in the nodeId[4], such that the bits at positions $b * l$ to $b * (l + 1) - 1$ represent level l. The value of the level defines a domain, where there are 2^b domains at each level, numbered from 0 to $2^b - 1$.

Each PAST node maintains a *routing table*. For each level l, the routing table contains the IP addresses of $2^b - 1$ nodes that have the same nodeId prefix as the present node up to level $l - 1$, but whose domains at level l have distinct values different from the corresponding domain in the present node's nodeId. How the information in the routing table is obtained is described later.

A node routes a message to a destination in the namespace by first finding the highest level l at which the node's id differs from the destination nodeId. The message is then forwarded either to a node that matches the destination nodeId up to at least level $l - 1$, or if no such node exists, to the node whose nodeId is numerically closest to the destination nodeId. This (slightly simplified) procedure guarantees that a message reaches the existing node whose nodeId is numerically closest to the destination nodeId in no more than $O(log_{2^b} N)$ steps, where N is the total number of PAST nodes.

An example is shown diagrammatically in Figure 1. The nodes are represented as circles and nodeIds. For simplicity, the nodeIds are limited to four bits, all the possible nodeIds exist, and $b = 1$. So, there are four levels, with the MSB of the nodeId corresponding to level zero. In the diagram, the domains for each level are shown. The shaded domains within each level highlight the domains that appear in the routing table of node 0110. Any node within a domain at a level can appear in the routing table of another node as a "representative" for that domain, so, in this example, any of the nodes 0000, 0001, 0010 or 0011 can act as the representative for the shaded domain zero at level one.

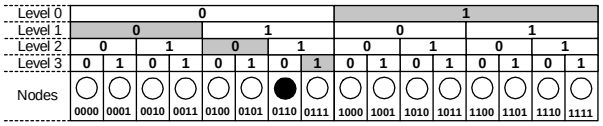

Figure 1: Tables maintained in each node.

To demonstrate the routing, if node 0110 receives a message with a destination of 0011, it will pass the message to one of nodes 0000, 0001, 0010 or 0011 (we will assume 0001 is in its routing table). This is because 0110 and 0011 differ at level one, and the routing table contains node 0001 for level one domain zero. Node 0001 will then route the message to either node 0011 or 0010, and so forth.

The number of entries in the routing table at each level is $2^b - 1$, and the number of levels that are populated in the routing table is $log_{2^b} N$. With a value of $b = 4$ and with as many as 10^{12} nodes, the routing table contains only approximately 150 entries and in the worst-case a message is routed through 10 nodes.

Each node also maintains two other tables, the *namespace table* and the *locality table*. The namespace table contains pointers to the K closest neighbours in the namespace. The locality table contains pointers to the N nodes that are "closest in the network" to the present node. Typical value for K and N, respectively, are 20 and 30. The namespace table is needed to support the routing in a sparsely populated space of nodeIds, and it increases the efficiency of document insertion. The role of the locality table will become clear later, when we discuss how the routing table is maintained.

Locality As described so far, the routing scheme does not take into account locality (in the network) at all. In reality, the "representative" node for each domain that appears in a node's routing table is not chosen randomly. Instead, the node that is "closest in the network" to the present node, among all nodes in a given domain (i.e., with a given nodeId prefix), is chosen as the representative. As a result, in each routing step, a query message is forwarded to the "closest" node that shares a longer (normally by one level) common nodeId prefix than the present node. This ensures that (1) a message is routed towards its' destination nodeId using a "good" (in terms of network proximity) route, and (2) a document query message will first reach the replicated copy of the requested document that is "closest" to the client.

Self-organising In order to be self-organising, PAST must be able to cope with the arrival of new nodes and the unannounced departure of other nodes. Due to space limitations, we limit our description of the self-configuration protocol to the case where a new node joins the system.

When a new node arrives, it needs to initialize its tables, and it needs to inform other nodes of its presence.

[4]Typically a value of 3 or 4 would be used for b.

We assume the new node knows initially about one other nearby (in the network) node A that is already part of the system. The new node asks the nearby node A to route a special "join" query message with a destination equal to the new node's proposed nodeId. Like any request, the message will be routed to the existing node B that is numerically closest to the proposed nodeId of the new node.

In response to receiving the "join" request, nodes A, B, and all nodes encountered on the path from A to B send their tables to the newly joining node. The new node inspects these tables and incorporates appropriate entries into its own tables. Furthermore, based on the tables received, the new node informs any nodes that need to be aware of the new node's arrival. We omit here some of the details, but one can show that this procedure ensures that the new node initializes its tables with the appropriate values, and that all other nodes that need to know about the new node's presence are notified and modify their tables appropriately.

Intuitively, this works because the new node receives appropriate values for the locality tables and the upper levels of the routing table from node A, which is close in the network. Similarly, it receives appropriate values for the namespace table and the lower levels of the routing tables from node B, whose nodeId is arithmetically close to the new node's id. Finally, appropriate values for the intermediate levels of the routing table are obtained from the nodes along the route from A to B.

Fault-tolerance The routing scheme as described so far is deterministic, and thus vulnerable to malicious or failed nodes along the route that accept messages but do not correctly forward them. Repeated queries could thus fail each time, since they are likely to take the same route.

To overcome this problem, the routing is actually randomized. To avoid routing loops, a message must always be forwarded to a node that is closer to the destination node in the namespace. However, the choice among multiple nodes that are all closer in the namespace to the destination is random. In practice, the probability distribution is heavily biased towards the best choice to ensure low average route delay. In the event of a malicious or failed node along the path, the query may have to be repeated several times by the client, until a route is chosen that avoids the bad node.

4 Related Work

There are currently many projects aimed at producing systems that provide peer-to-peer style interaction, and in particular systems that provide large-scale storing and sharing of documents.

Some of the systems can be classified as providing sharing facilities, such as Gnutella [3], Freenet [2], Napster [1]. Whilst these systems have proved popular, they provide only weak persistence, may suffer from limited scalability (e.g. Gnutella), or use centralised components (e.g. Napster). PAST shares with these systems the use of untrusted nodes and the anonymity aspects (e.g. FreeNet), but combines these properties with strong persistence and scalability.

There are a number of systems that aim to provide high availability and persistence, such as Oceanstore [6], FarSite [4], Publius [7] and implementations of Eternity [5]. Many of these systems attempt to provide traditional file system type semantics, supporting mutable data, directory structures and so forth. PAST differs from these projects in its combination of immutable document storage, scalability, anonymity, strong persistence, untrusted nodes, self-configuration and the use of a broker.

PAST's routing algorithm bears some similarity to Plaxton trees [8]. However, there are important differences. In PAST, there is no single "document root", which forms a single point of failure. Also, the way locality is achieved is completely different in PAST and unlike Plaxton trees, PAST normally ensures that a document query is routed to the live replica closest to the client.

5 Status and Conclusion

We currently have a working prototype of the system that operates in a simulated network environment. We have performed tests with up to 10,000 PAST nodes and more then a million documents. Self-configuration and document storage/retrieval are fully functional, and early performance results have been very encouraging.

Plans for the immediate future are to perform more extensive simulations, to verify PAST's security model more formally, and to complete and distribute an implementation that can be deployed in the Internet. A full-length paper with a detailed description of the system along with an experimental evaluation is forthcoming.

In this paper, we have given a brief overview of the design of PAST, an Internet based peer-to-peer scalable, persistent and anonymous storage utility. A key feature of PAST highlighted in this paper is the use of a broker, which issues smartcards that act as the system's TCB. We argued that this feature allows PAST to provide strong assurances, despite weak assumptions about the behaviour and trustworthiness of individual nodes. We hope that the paper will help stimulate debate about the use of a broker in peer-to-peer systems such as PAST.

References

[1] Napster. http://www.napster.com/.

[2] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, July 2000. ICSI, Berkeley, CA, USA.

[3] The Gnutella protocol specification, 2000. http://dss.clip2.com/GnutellaProtocol04.pdf.

[4] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed

on an existing set of desktop pcs. In *Proc. SIGMETRICS'2000*, pages 34–43, 2000.

[5] R.J. Anderson. The eternity service. In *Proc. PRAGOCRYPT'96*, pages 242–252. CTU Publishing House, 1996. Prague, Czech Republic.

[6] John Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent store. In *Proc. ASPLOS'2000*, November 2000.

[7] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[8] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, pages 311–320, June 1997. Newport, Rhode Island, USA.