Enforcing Fair Sharing of Peer-to-Peer Resources

Tsuen-Wan “Johnny” Ngan, Dan S. Wallach, and Peter Druschel
Department of Computer Science, Rice University
{twngan,dwallach,druschel}@cs.rice.edu

Abstract

Cooperative peer-to-peer applications are designed to share the resources of each computer in an overlay network for the common good of everyone. However, users do not necessarily have an incentive to donate resources to the system if they can get the system’s resources for free. This paper presents architectures for fair sharing of storage resources that are robust against collusions among nodes. We show how requiring nodes to publish auditable records of their usage can give nodes economic incentives to report their usage truthfully, and we present simulation results that show the communication overhead of auditing is small and scales well to large networks.

1 Introduction

A large number of peer-to-peer (p2p) systems have been developed recently, providing a general-purpose network substrate [10, 11, 13, 14, 16] suitable for sharing files [6, 7], among other applications. In practice, particularly with widespread p2p systems such as Napster, Gnutella, or Kazaa, many users may choose to consume the p2p system’s resources without providing any of their own resources for the use of others [1]. Users have no natural *incentive* to provide services to their peers if it is not somehow required of them.

This paper considers methods to design such requirements directly into the p2p system. While we could take a traditional quota enforcement approach, requiring some kind of trusted authority to give a user “permission” to store files, such notions are hard to create in a network of peers. Why should some peers be placed in a position of authority over others? If all nodes were to publish their resource usage records, directly, where other nodes are auditing those records as a part of the normal functioning of the system, we might be able to create a system where nodes have natural incentives to publish their records accurately. Ideally, we would like to design a system where nodes, acting selfishly, behave collectively to maximize the common welfare. When such a system has no centralized authority with total knowledge of the system making decisions, this becomes a distributed algorithmic mechanism design (DAMD) problem [9], a current area of study which combines computational tractability in theoretical computer science with incentive-compatible mechanism design in the economics literature.

To illustrate the power of such economic systems, we focus on the specific problem of fair sharing in p2p storage systems, although our techniques can potentially be extended to discuss fairness in bandwidth consumption and other resources. Section 2 discusses adversarial models that a storage system must be designed to address. Section 3 discusses different approaches to implementing fairness policies in p2p storage systems. Section 4 presents some simulation results. Finally, Section 5 discusses related work and Section 6 concludes.

2 Models

Our goal is to support a notion of fair sharing such as limiting any given node to only consuming as much of the network’s storage as it provides space for others on its local disk. A centralized broker that monitored all transactions could accomplish such a feat, but it would not easily scale to large numbers of nodes, and it would form a single point of failure; if the broker was offline, all file storage operations would be unable to proceed.

We will discuss several possible decentralized designs in Section 3, where nodes in the p2p network keep track of each others’ usage, but first we need to understand the threats such a design must address. It is possible that some nodes may wish to collude to corrupt the system, perhaps gaining more storage for each other than they collectively provide to the network. We consider three adversarial models:

No collusion Nodes, acting on their own, wish to gain an unfair advantage over the network, but they have no peers with which to collude.

Minority collusion A subset of the p2p network is willing to form a conspiracy to lie about their resource usage. However, it is assumed that most nodes in the p2p network are uninterested in joining the conspiracy.

Minority bribery The adversary may choose specific nodes to join the conspiracy, perhaps offering them a bribe in the form of unfairly increased resource usage.

This paper focuses primarily on minority collusions. While bribery is perfectly feasible, and we may well even be able to build mechanisms that are robust against bribery, it is entirely unclear that the lower-level p2p routing and messaging systems can be equally robust. In studying routing security for p2p systems, Castro et al. [3] focused only on minority collusions. Minority bribery would allow very small conspiracies of nodes to defeat the secure routing primitives. For the remainder of this paper, we assume the correctness of the underlying p2p system.

We note that the ability to consume resources, such as remote disk storage, is a form of currency, where remote resources have more value to a node than its local storage. When nodes exchange their local storage for others' remote storage, the trade benefits both parties, giving an incentive for them to cooperate. As such, there is no need for cash or other forms of money to exchange hands; the storage economy can be expressed strictly as a barter economy.

3 Designs

In this section, we describe three possible designs for storage accounting systems. For all of these designs, we assume the existence of a public key infrastructure, allowing any node to digitally sign a document such that any other node can verify it, yet it is computationally infeasible for others to forge.

Likewise, for any of these designs, it is imperative to ensure that nodes are actually storing the files they claim to store. This is guaranteed by the following *challenge* mechanism. For each file a node is storing, it periodically picks a node that stores a replica of the same file as a target, and notifies all other replicas holders of the file that it is challenging that target. Then it randomly selects a few blocks of the file and queries the target for the hash of those blocks. The target can answer correctly only if it has the file. The target may ask another replica holder for a copy of the file, but any such request during a challenge would cause the challenger to be notified, and thus able to restart the challenge for another file.

3.1 Smart cards

The original PAST paper [7] suggested the use of smart cards to enforce storage quotas. The smart card produces signed endorsements of a node's requests to consume remote storage, while charging that space to an internal counter. When storage is reclaimed, the remote node returns a signed message that the smart card can verify before crediting its internal counter.

Smart cards avoid the bandwidth overheads of the decentralized designs discussed in this paper. However, smart cards must be issued by a trusted organization, and periodically re-issued to invalidate compromised cards. This requires a business model that generates revenues to cover the cost of running the organization. Thus, smart cards appear to be unsuitable for grassroots p2p systems.

3.2 Quota managers

If each smart card was replaced by a collection of nodes in the p2p network, the same design would still be applicable. We can define the *manager set* for a node to be a set of nodes adjacent to that node in the overlays node indentifier (nodeId) space, making them easy for other parties in the overlay to discover and verify. Each manager must remember the amount of storage consumed by the nodes it manages and must endorse all requests from the managed nodes to store new files. To be robust against minority collusion, a remote node would insist that a majority of the manager nodes agree that a given request is authorized, requiring the manager set to perform a Byzantine agreement protocol [4].

The drawback of this design is that request approval has a relatively high latency and the number of malicous nodes in any manager set must be less than one third of the set size. Furthermore, managers suffer no direct penalty if they grant requests that would be correctly denied, and thus could be vulnerable to bribery attacks.

3.3 Auditing

While the smart card and quota manager designs are focused on enforcing quotas, an alternative approach is to require nodes to maintain their own records and publish them, such that other nodes can audit those records. Of course, nodes have no inherent reason to publish their records accurately. This subsection describes how we can create natural economic disincentives to nodes lying in their records.

3.3.1 Usage files

Every node maintains a *usage file*, digitally signed, which is available for any other node to read. The usage file has three sections:

- the *advertised capacity* this node is providing to the system;
- a *local list* of (nodeId, fileId) pairs, containing the identifiers and sizes of all files that the node is storing locally on behalf of other nodes; and
- a *remote list* of fileIds of all the files published by this node (stored remotely), with their sizes.

Together, the local and remote lists describe all the credits and debits to a node's account. Note that the nodeIds for the peers storing the files are not stored in the remote list, since this information can be found using mechanisms in the storage system (e.g., PAST). We say a node is "under quota," and thus allowed to write new files into the network, when its advertised capacity minus the sum of its remote list, charging for each replica, is positive.

When a node A wishes to store a file F_1 on another node B, first B must fetch A's usage file to verify that A is under quota. Then, two records are created: A adds F_1 to its remote list and B adds (A, F_1) to its local list. This is illustrated in Figure 1. Of course, A might fabricate the contents of its usage file to convince B to improperly accept its files.

We must provide incentives for A to tell the truth. To game the system, A might normally attempt to either *inflate* its advertised capacity or *deflate* the sum of its remote list. If A were to increase its advertised capacity beyond the amount of disk it actually has, this might attract storage requests that A cannot honor, assuming the p2p storage system is operating at or near capacity, which is probably a safe assumption. A might compensate by creating fraudulent entries in its local list, to claim the storage is being used. To prevent fraudulent entries in either list, we define an auditing procedure that B, or any other node, may perform on A.

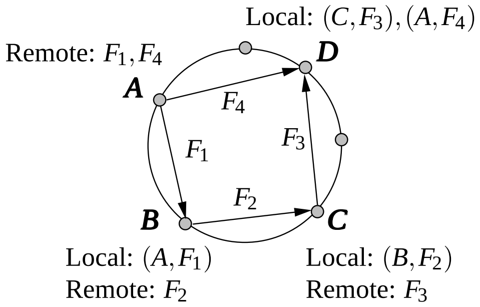

Figure 1: A p2p network with local/remote lists.

If B detects that F_1 is missing from A's remote list, then B can feel free to delete the file. After all, A is no longer "paying" for it. Because an audit could be gamed if A knew the identity of its auditor, anonymous communication is required, and can be accomplished using a technique similar to Crowds [12]. So long as every node that has a relationship with A is auditing it at randomly chosen intervals, A cannot distinguish whether it is being audited by B or any other node with files in its remote list. We refer to this process as a *normal audit*.

Normal auditing, alone, does not provide a disincentive to inflation of the local list. For every entry in A's local list, there should exist an entry for that file in another node's remote list. An auditor could fetch the usage file from A and then connect to every node mentioned in A's local list to test for matching entries. This would detect inconsistencies in A's usage file, but A could collude with other nodes to push its debts off its own books. To fully audit A, the auditor would need to audit the nodes reachable from A's local list, and recursively audit the nodes reachable from those local lists. Eventually, the audit would discover a *cheating anchor* where the books did not balance (see Figure 2). Implementing such a recursive audit would be prohibitively expensive. Instead, we require all nodes in the p2p overlay to perform *random auditing*. With a lower frequency than their normal audits, each node should choose a node at random from the p2p overlay. The auditor fetches the usage file, and verifies it against the nodes mentioned in that file's local list. Assuming all nodes perform these random audits on a regular schedule, every node will be audited, on a regular

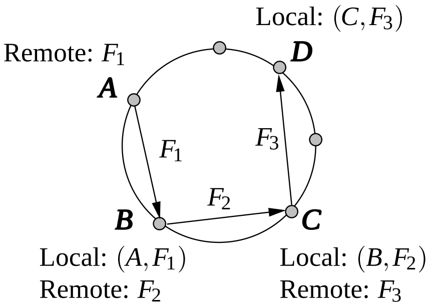

Figure 2: A cheating chain, where node A is the cheating anchor.

basis, with high probability.

How high? Consider a network with n nodes, where $c < n$ nodes are conspiring. The probability that the cheating anchor is not random audited by any node in one period is $\left(\frac{n-2}{n-1}\right)^{n-c} > 1/e \approx 0.368$, and the cheating anchor would be discovered in three periods with probability higher than 95%.

Recall that usage files are digitally signed by their node. Once a cheating anchor has been discovered, its usage file is effectively a *signed confession* of its misbehavior! This confession can be presented as evidence toward ejecting the cheater from the p2p network. With the cheating anchor ejected, other cheaters who depended on the cheating anchor will now be exposed and subject to ejection, themselves.

We note that this design is robust even against bribery attacks, because the collusion will still be discovered and the cheaters ejected. We also note that since everybody, including auditors, benefits when cheaters are discovered and ejected from the p2p network, nodes do have an incentive to perform these random audits [8].

3.3.2 Extensions

Selling overcapacity As described above, a node cannot consume more resources from the network than it provides itself. However, it is easy to imagine nodes who want to consume more resources than they provide, and, likewise, nodes who provide more resources than they wish to consume. Naturally, this overcapacity could be sold, perhaps through an online bidding system [5], for real-world money. These trades could be directly indicated in the local and remote lists. For example, if D sells 1GB to E, D can write (E, 1GB trade) in its remote list, and E writes (D, 1GB trade) in its local list. All the auditing mechanisms continue to function.

Reducing communication Another issue is that fetching usage logs repeatedly could result in serious communication overhead, particularly for nodes with slow net connections. To address this, we implemented three optimizations. First, rather than sending the usage logs through the overlay route used to reach it, they can be sent directly over the Internet: one hop from the target node to the anonymizing relay, and one hop to the auditing node. Second, since an entry in a remote list would be audited by all nodes replicating the logs, those replicas can alternately audit that node to share the cost of auditing. Third, we can reduce communication by only transmitting diffs of usage logs, since the logs change slowly. We must be careful that the anonymity of auditors isn't compromised, perhaps using version numbers to act as cookies to track auditors. To address this, the auditor needs to, with some probability, request the complete usage logs.

4 Experiments

In this section, we present some simulation results of the communication costs of the quota managers and the auditing system. For our simulations, we assume all nodes are following the rules and no nodes are cheating. Both storage space and file sizes are chosen from truncated normal distributions[1]. The storage space of each node is chosen from 2 to 200GB, with an average of 48GB. We varied the average file size across experiments. In each day of simulated time, 1% of the files are reclaimed and republished. Two challenges are made to random replicas per file a node is storing per day.

For quota managers, we implemented Castro et al.'s BFT algorithm [4]. With a manager set size of ten, the protocol can tolerate three nodes with byzantine faults in any manager set. For auditing, normal audits are performed on average four times daily on each entry in a node's remote list and random audits are done once per day. We simulated both with and without the append-only log optimization.

Our simulations include per-node overhead for

[1] The bandwidth consumed for auditing is dependent on the number, rather than the size, of files being stored. We also performed simulations using heavy-tailed file size distributions and obtained similar results.

Pastry-style routing lookups as well as choosing one node, at random, to create one level of indirection on audit requests. The latter provides weak anonymity sufficient for our purposes. Note that we only measure the communication overhead due to storage accounting. In particular, we exclude the cost of p2p overlay maintenance and storing/fetching of files, since it is not relevant to our comparison. Unless otherwise specified, all simulations are done with 10,000 nodes, 285 files stored per nodes, and an average node lifetime of 14 days.

4.1 Results

Figure 3 shows the average upstream bandwidth required per node, as a function of the number of nodes (the average required downstream bandwidth is identical). The per-node bandwidth requirement is almost constant, thus all systems scale well with the size of the overlay network.

Figure 4 shows the bandwidth requirement as a function of the number of files stored per node. The overheads grow linearly with the number of files, but for auditing without caching, it grows nearly twice as fast as the other two designs. Since p2p storage systems are typically used to store large files, this overhead is not a concern. Also, the system could charge for an appropriate minimum file size to give users an incentive to combine small files into larger archives prior to storing them.

Figure 5 shown the overhead versus average node lifetime. The overhead for quota managers grows rapidly when the node lifetime gets shorter, mostly from the cost in joining and leaving manager sets and from voting for file insertions for new nodes.

Our simulations have also shown that quota managers are more affected by the file turnover rate, due to the higher cost for voting. Also, the size of manager sets determines the vulnerability of the quota manager design. To tolerate more malicious nodes, we need to increase the size of manager sets, which would result in a higher cost.

In summary, auditing with caching has performance comparable to quota managers, but is not subject to bribery attacks and is less sensitive to the fraction of malicious nodes. Furthermore, in a variety of conditions, the auditing overhead is quite low — only a fraction of a typical p2p node's bandwidth.

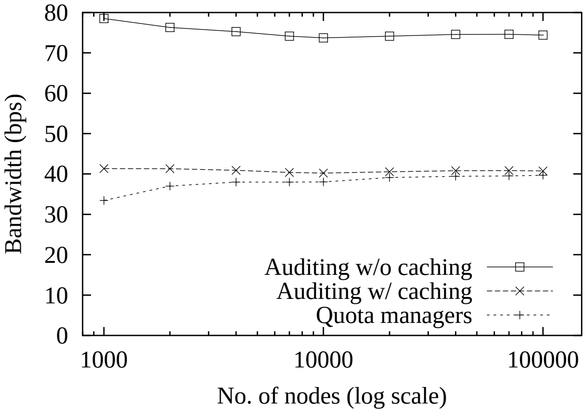

Figure 3: Overhead with different number of nodes.

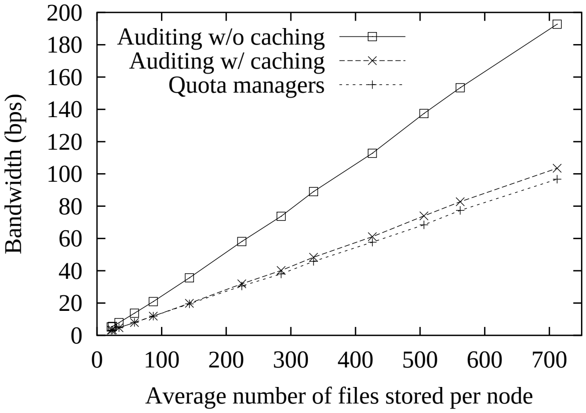

Figure 4: Overhead with different number of files stored per node.

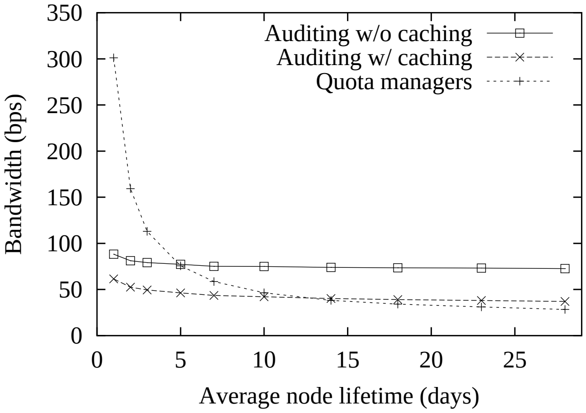

Figure 5: Overhead with different average node lifetime.

5 Related Work

Tangler [15] is designed to provide censorship-resistant publication over a small number of servers (i.e., < 30), exchanging data frequently with one another. To maintain fairness, Tangler requires servers to obtain "certificates" from other servers which can be redeemed to publish files for a limited time. A new server can only obtain these certificates by providing storage for the use of other servers and is not allowed to publish anything for its first month online. As such, new servers must have demonstrated good service to the p2p network before being allowed to consume any network services.

The Eternity Service [2] includes an explicit notion of electronic cash, with which users can purchase storage space. Once published, a document cannot be deleted, even if requested by the publisher.

Fehr and Gachter's study considered an economic game where selfishness was feasible but could easily be detected [8]. When their human test subjects were given the opportunity to spend their money to punish selfish peers, they did so, resulting in a system with less selfish behaviors. This result helps justify that users will be willing to pay the costs of random audits.

6 Conclusions

This paper has presented two architectures for achieving fair sharing of resources in p2p networks. Experimental results indicate small overheads and scalability to large numbers of files and nodes. In practice, auditing provides incentives, allowing us to benefit from its increased resistance to collusion and bribery attacks.

Acknowledgments

We thank Moez A. Abdel-Gawad, Shu Du, and Khaled Elmeleegy for their work on an earlier version of the quota managers design. We also thank Andrew Fuqua and Hervé Moulin for helpful discussions on economic incentives. This research was supported by NSF grant CCR-9985332, Texas ATP grants 003604-0053-2001 and 003604-0079-2001, and a gift from Microsoft Research.

References

[1] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), Oct. 2000.

[2] R. Anderson. The Eternity service. In *Proc. 1st Int'l Conf. on the Theory and Applications of Cryptology*, pages 242–252, Prague, Czech Republic, Oct. 1996.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. OSDI'02*, Boston, MA, Dec. 2002.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. OSDI'99*, New Orleans, LA, Feb. 1999.

[5] B. F. Cooper and H. Garcia-Molina. Bidding for storage space in a peer-to-peer data preservation system. In *Proc. 22nd Int'l Conf. on Distributed Computing Systems*, Vienna, Austria, July 2002.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. SOSP'01*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[7] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. 8th Workshop on Hot Topics in Operating Systems*, Schoss Elmau, Germany, May 2001.

[8] E. Fehr and S. Gachter. Altruistic punishment in humans. *Nature*, (415):137–140, Jan. 2002.

[9] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. 6th Int'l Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, Atlanta, GA, Sept. 2002.

[10] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. IPTPS'02*, Cambridge, MA, Mar. 2002.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM'01*, pages 161–172, San Diego, CA, Aug. 2001.

[12] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object address and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, Nov. 2001.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. SIGCOMM'01*, San Diego, CA, Aug. 2001.

[15] M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proc. 8th ACM Conf. on Computer and Communications Security*, Nov. 2001.

[16] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area address and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, Apr. 2001.