Enforcing Fair Sharing of Peer-to-Peer Resources

Written by

Stefan Chouteau

Content

1. Introduction

In the recent years, there have been a large number of peer-to-peer systems for a lot of purposes. Some of them have been suitable for sharing files, like Napster, Gnutella or Kazaa, just to name some of them. One of the great Problems of those systems has been that users had no natural incentive to provide services to the system. Why provide my own resources, when I can get the other ones for free?

Those selfish people are often called “leecher”. You probably can imagine that that leecher’s can cause serious problems in a peer-to-peer system. If there are more people who want to use the systems resources than people who are willing to provide their own resources to the system, there won’t be a good performance.

You will learn about different models, which create incentives (for the peers) to provide resources to the p2p-system, which will be directly implemented into the p2p-system. To introduce these models, we will use a fictitious p2p storage system, which I’m going to explain later in more detail.

2. Models of design, a p2p storage system must be designed to address to

The goal we want to reach is a notion of fair sharing. In other words every peer in the p2p-system shall only be able to consume as much of the systems resources, than it provides itself to the system. In a storage system, this means every node will only be able to consume as much remote storage, than it provides space for others on its own local disk.

But how to guarantee that a peer will only use as much resources as it provides?

In a peer-to-peer system, you need a decentralized approach, which guarantees that all peers are still equal, and no peer takes a position of greater authority over others than anyone else. In Chapter 3, you will see two possible decentralized designs, which are able to monitor the transactions of every peer and that fulfil this notion.

Another advantage of a decentralized approach is that there is no single point of failure, it is more robust to failures than a centralized approach and it will easily scale to large numbers of nodes, but more on this later.

Let’s first try to understand the …

2.1 Threats a decentralized Design must address

It is possible that some nodes which to collude to corrupt the system, perhaps trying to gain more storage for each other, than they collectively provide to the network. We differentiate between three kinds of corruption:

- No Collusion
- Minority Collusion
- Minority Bribery

No Collusion means there are some nodes in the system, acting with the purpose of gaining an unfair advantage over the network, but they are acting on their own, because they have no one to collude.

Minority Collusion means a subset of the p2p-system is willing to form a conspiracy to lie about their resources, but it is assumed that most nodes in the p2p-system are uninterested in joining the conspiracy.

Minority Bribery is some kind of advanced collusion, there is still a subset of nodes, which want to cheat on the others, but in this case, those nodes try to bribe other nodes, perhaps by offering them to lie on their resource usage, with the intention to get this node's joining the conspiracy.

It is perfectly feasible that there will be bribery in such a system, and we may even been able to build mechanisms against bribery, but it is entirely unclear that the lower level p2p routing and messaging systems can be equally robust. For the rest of the paper, we assume the correctness of the underlying p2p system.

2.2 Incentives in a p2p storage system

Why shall peers in our storage system provide their local storage to others? What incentive shall they have to do so?

In a p2p storage system, the ability to consume resources can be seen as some kind of currency. In such a system, it is just feasible that remote storage is more valuable to a node than its local storage. When now a node exchanges its local storage against another nodes remote storage both parties will benefit of the trade, giving them an incentive to cooperate. One probably should mention that such a storage economy can be expressed strictly as a barter economy; there is no need for money or other forms of payment to exchange hands.

3. Different approaches of implementing fairness policies in p2p storage systems

In this chapter, you will see some different approaches of implementing fairness policies in our storage system. At first, we have to make two assumptions our system will need to work properly.

At first, we need a public key infrastructure that allows nodes to digitally sign documents, so other nodes can verify it. It shall be computationally infeasible to forge.

Second, it is imperative to any of these implementations that nodes are actually storing the files they claim to store. Because of that, we need a challenge mechanism. This challenge mechanism picks for each file a node is storing a node, which is holding a replica of that file. This happens in periodical intervals. After that, the challenging node informs all other replica holders that it is going to challenge its target. Then it randomly selects a few blocks of the file and queries the target for the hash of those blocks. The target can only answer this query if it has the file. If it hasn't, it will probably ask a replica holder for the file, but any such request during the challenge would cause the challenger to be informed and because of that be able to restart the challenge for another file.

3.1 Quota Approach

One of the first Quota approaches in p2p storage systems has been mentioned in PAST. PAST is an internet based p2p global storage utility which aims to provide strong persistence, high availability, scalability and security. The PAST Paper suggested the use of smart cards that produce signed endorsements of a node’s request to consume remote storage. The consumed space is charged to an internal counter. If storage is reclaimed, the counter will be credited.

But such a smart card approach has a lot of disadvantages. You need a trusted organization that issues the cards. After a period of time the smartcards will have been to re-issue to invalidate compromised cards. This will raise costs that someone will have to pay. Therefore one needs a business model to cover the costs. You see, this seems to be unsuitable for a grassroots p2p system.

How can we improve this notion?

If each smart card would be replaced by a set of nodes, the same design would be applicable. This set is called the *manager set* for a node. It is defined to be a set of nodes adjacent to that node in the overlays node ID space. This makes it easier for other parties in the overlay to discover and verify them. The Job of those managers is to remember the amount of storage consumed by the nodes they manage and to endorse all requests from the managed nodes to store new files.

To make this system robust against minority collusion, a remote node would insist that a majority of the manager nodes agree that a given request is authorized. This requires the manager set to perform a Byzantine Agreement protocol.

The protocol works roughly as follows:

1. A node A asks the manager set for node B if B is allowed to store new files in node A’s remote storage
2. Each node in the manager set checks its records and sends a reply
3. Node A waits for a certain amount of same answers before the storage process is granted or denied

The drawback of this design is that request approval causes relatively high latency. The number of malicious nodes must be less than a certain percentage in any manager set to guarantee that the approval process couldn’t be disturbed and managers suffer no penalty if they grant requests that would be correctly denied.

This means, this design is Vulnerable to bribery attacks.

3.2 Auditing

Let’s take a closer look on another approach, Auditing. Unlike the quota manager design, nodes are required to maintain and publish their own usage records, such that other nodes can audit them. Of course nodes have no natural incentive to tell the truth about their records. Because of that we have to create disincentives to nodes lying on their records.

Let’s first take a closer look on the usage records, the so called usage file. Every node maintains a usage file, which is digitally signed and available for any other node to verify. The usage file consists of three sections:

- The advertised capacity
- The local list
- The remote list

The advertised capacity is the amount of disk space, a user provides to the system.

The local list consists of (node ID, file ID) pairs, containing the identifiers and sizes of all files that the node is storing on its local disk on behalf of others.

The remote list consists only of the file ID’s of all files published by this node. Node ID’s aren’t necessary, cause this information can be found using mechanisms in the storage system.

Together the local and remote list describe all the credits and debits to a node’s account. We say a node is “under quota” when its advertised capacity minus the sum of its remote list, charging for each replica, is positive.

Concerning the usage file, there are two possibilities for a node to cheat on others. A node could either inflate its advertised capacity beyond the disk space it really has or deflate the sum of its remote list.

When increasing its advertised capacity beyond the resources of the disk, this might attract storage requests that the node cannot honour. The node may try to compensate by creating fraudulent entries in its local list, to claim the storage is being used.

The second possibility is to deflate the remote list. This can be done by just deleting entries without informing the appropriate node that he can delete the file.

To prevent nodes from cheating, we need auditing procedures that any node can perform on others. These procedures are called:

- Normal audit
- Random audit

When doing a normal audit, a node detects for any file in its local list, if there is an entry in the appropriate node's remote list. If the entry is missing, the auditing node can feel free to delete the file, cause no one is paying anymore for it. It is essential that the auditing is anonymous and done at randomly chosen intervals. If the audited node could distinguish its auditor, the entry could be restored for the time of auditing, so the audit would only be gamed. This mechanism prevents fraudulent entries in a node's remote list.

When doing a random audit, a node checks for another node's local list. For every entry in this local list, there should be an appropriate entry in another node's remote list. If no entry can be found, the node, which had signed the correctness of his books, and whose books imbalance will be ejected by the system.

This procedure is even robust against bribery. If there is bribery, and one of those cheating nodes, who form a so called cheating chain, is discovered and ejected, it will only be a matter of time until the other cheating nodes will be ejected too.

We require all nodes to perform random auditing with a lower frequency than their normal audits. Each node should choose a node at random from the overlay network. Assuming all nodes perform these random audits on a regular schedule, every node will be audited on a regular basis with high probability.

3.3 Extensions

As already mentioned, a node cannot consume more resources than it provides, but it is easy to imagine nodes that want to consume more resources than they provide or that provide more resources than they consume. This overcapacity could be sold, perhaps through a online bidding system for real-world money. These Trades could be directly indicated in the local and remote lists, using entries like

(Node ID, Amount Trade) for example, where the selling node writes the entry in its remote list and the buying node writes the entry in its local list.

Another improvement that could be done is concerning the usage files.

Fetching those files repeatedly could result in serious communication overhead. We could implement some improvements to reduce this overhead, eventually sending the usage files directly through the internet, using an anonymizing relay, instead of using the overlay network. This would result in a route of only two hops from the source to the anonymising relay to the target.

Another way to reduce communication is to let the replica holders of a file audit the publishing node alternately.

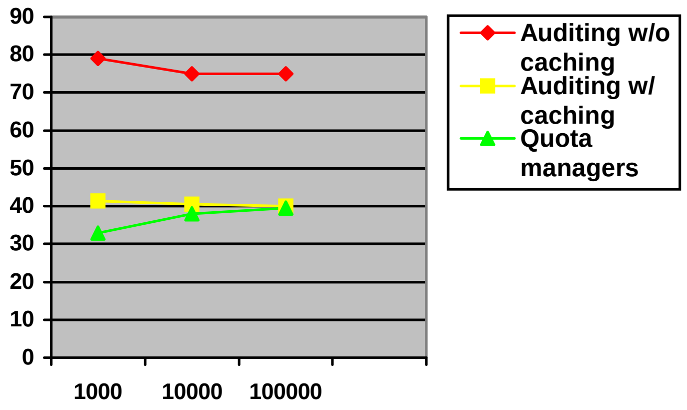
Or, last but not least only transmit the differences of usage files, because those files will probably change slowly.

4. Simulation results concerning bandwidth overhead

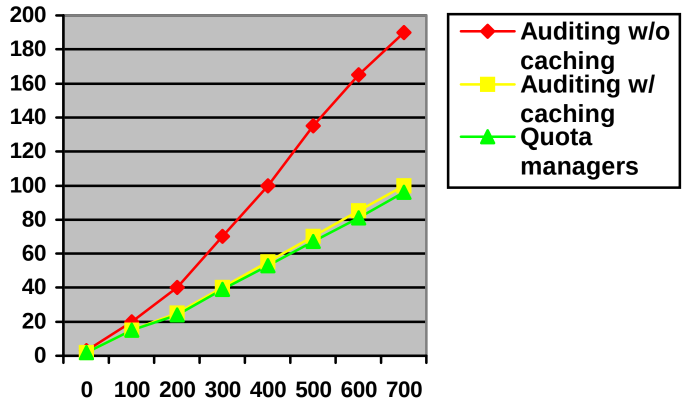
In this chapter, you will see some simulation results, concerning the bandwidth overhead of Quota Managers, auditing without and auditing with caching. Caching means only the differences of usage files are transmitted.

For the simulation, we assume all nodes are following the rules and no nodes are cheating. The storage space of each node is chosen from 2 Gigabyte up to 200 Gigabyte with an average of 48 Gigabyte. 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, the manager set size is ten. For Auditing, normal audits are performed on average four times daily on each entry in a nodes remote list, random audits are done once per day

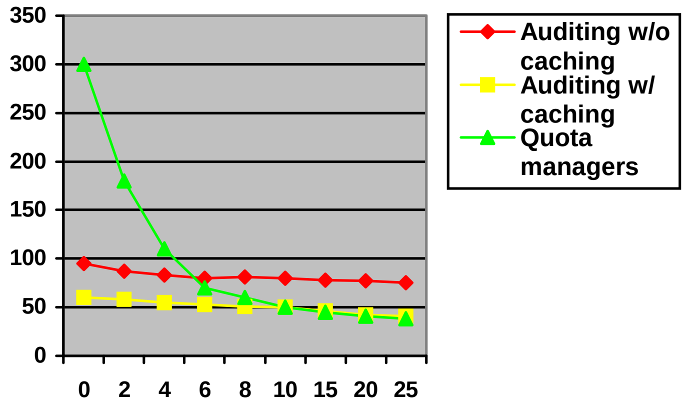
Unless otherwise specified, all simulations are done with 10.000 Nodes, 285 files per node and an average node lifetime of 14 days.



4.1 Overhead with different number of nodes



4.2 Overhead with different number of files per node



4.3 Overhead with different average node lifetime

As you can see, the auditing overhead is quite low – only a fraction of a typical p2p node's bandwidth. 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. Quota Managers get mostly affected by the average node lifetime.

5. Personal Conclusion

You’ve seen two different approaches of enforcing fair sharing directly into the p2p system. One based on requests and agreements, one based on self-maintenance and audits. Each of these approaches has its advantages and disadvantages. However, to me, auditing with caching seems to be the best solution of all, its robust against bribery and has nearly the same low bandwidth overhead as quota managers have.

At least, enforcing fair sharing of peer-to-peer resources is possible, it can be implemented directly into the p2p system and it can be even robust against bribery which leads to the benefit of all.