Byzantine Federated Storage

David Vorick
Nebulous Labs

September 14, 2014

Abstract

We explore a method of randomly distributing files to hosts in a dynamic system, in a way that prevents malicious hosts from manipulating the distribution of files on the network under certain constraints.

1 Introduction

We wish to build a network of hosts and files, where every host stores a set of files, and every file is stored by exactly one host. This network is dynamic, meaning that files can be added at any time, and hosts can be added at any time. We further wish to perform consensus operations on this network. To prevent Sybil attacks, hosts must submit storage proofs to be eligible to participate in consensus. In order to protect consensus, we only wish to make sure that no malicious party can gain control of a majority of the voting power.

We make the following assumptions:

- All non-malicious files are fully compressed and non-redundant.
- A mechanism for proof of storage exists.
- No party can afford more than 50% of the total resources available to the network. That is, a party cannot perform a 50% attack by buying enough storage hardware, renting 50% of the storage available on the network, or some combination of the two.
- Renting a volume of storage on the network is at least as expensive as renting the same volume off the network.

Given these assumptions, we wish to build a network that satisfies the following properties:

- Every file is stored on a host.
- On a sufficiently large network, the varience in the number of files stored by each host is bounded.
- A party controlling less than 50% of the physical storage on the network cannot appear to control greater than 50% of the storage without incurring greater economic cost than the cost of owning 50% of the storage.

We establish that the following condition is sufficient to achieve the third property:

- A set of hosts that comprise less than 50% of the network cannot probabilistically control greater than 50% of the files they have uploaded at any time without incurring prohibitive economic cost.

Because we assume a mechanism for proof of storage, we hold that the only way for a party to appear to control greater storage than it actually controls is to through the use of a specially constructed "fake file" whose storage proof can be spoofed. This allows the malicious party to pretend to store the file without needing to consume physical storage. To prevent such attacks, we must ensure that the cost of spoofing the storage proof is more expensive than the cost of genuinely storing the file, which is what we will define as prohibitive economic cost.

The fake file attack requires that the fake file uploaded to the network land on a conspiring host. If the fake file has a greater chance of being on a non-conspiring host than a conspiring host, the cost of keeping the fake file will exceed the cost required to just buy raw storage. If under no circumstances can a set of hosts comprising less than 50% of the hosts on the network probabilistically be responsible for hosting greater than 50% of the fake files they have uploaded without incurring prohibitive economic cost, then the conspiring hosts will never be able to appear as though they control greater than 50% of the raw storage space on the network.

The remainder of the paper shall design a system that prevents such an attack. The following properties will be achieved (given the prior assumptions):

1. Every file is stored on a host.
2. A set of hosts that comprise less than 50% of the network cannot probabilistically control greater than 50% of the files they have uploaded at any time without incurring prohibitive economic cost.
3. On a sufficiently large network, the variance in the number of files stored by each host is bounded.

Throughout this paper, only the quantity of files is discussed, with no mention of how large these files might be. It can be assumed that all files are given a weight equal to their size in bytes when calculating penalties or making other considerations.

2 A Non-Scaling Solution

A modified rendezvouz hashing scheme will be used to achieve the aforementioned goals. Every host and file on the network is paired with a trusted random seed after joining the network. The source of these seeds is outside the scope of this paper, though something like the hash of a future Bitcoin block could be used. To determine which host stores a file, each host is assigned a number determined by concatenating its seed with the seed of file and hashing the result. The host with the largest hash is chosen to store the file.

To achieve the first property, we simply require that at least one host be on the network at all times.

Achieveing the second property requires enumerating all of the ways in which the network can reconfigure, and then demonstrating that non of the reconfigurations are vulnerable to manipulation. The network will reconfigure during the following actions:

- A file is added to the network.
- A file is removed from the network.
- A host is added to the network.
- A host is removed from the network.

We can keep things easy by enforcing that only one of these actions can happen at a time.

2.1 Adding and Removing Files

When a file is added to the network, it is given a random seed which will result in it being put onto a random host in the network. This process is sufficiently random, unless a conspirator tries to 'reroll' the seed of a file by uploading the file, then having the file taken off of the network, and then re-uploading the file. We limit this by forcing files to be pre-paid, preventing funds from being added to a file after it has been uploaded, and preventing a file from being removed from the network until the funds are gone. We also prevent the contents of the file from being altered, so that a conspiring uploader cannot edit a file to be 'fake' after uploading and seeing whether the file lands in a desirable location. With these limitations, there is no way for an uploader to manipulate the probability of a file landing on a conspiring host.

Files can be removed from the network prematurely if the host that was storing the file loses the file. This is the only way that files can be prematurely removed from the network. When this occurs, the uploader is refunded for the remaining time that the file could have spent on the network. A conspiring host can potentially corrupt files it is supposed to be storing in order to manipulate the portion of fake files that it is storing, to increase that portion above 50%, violating the second condition. The act of corrupting a file removes a host from the responsibility of storing that file into the future (and also removes the responsibility of any successor the host may have from storing the file into the future). Therefore, a penalty must be incurred on the host that is as economically expensive as holding onto the file for the duration of the file's life. This is as simple as forcing the host to pay for the time remaining value in the file, taking the place of the original uploader for the expense.

Using this penalty has a nice property: it allows hosts to selectively corrupt files without having to leave the network entirely. This is good for legal reasons, as it allows hosts under legal pressure to delete files that are illegal without being forced away from participation in the network. This may satisfy potential participants who are concerned about the legal and moral responsibilities accompanying hosting potentially objectionable files. It is also noted that this property does not eliminate privacy: through encryption and obfuscation, an uploader can protect their own privacy, while still allowing law enforcement to prevent the distribution of illegal content without disrupting the foundational functioning of the network.

When a host joins the network, it receives a random set of files from throughout the network, taking them from random places. The host can manipulate the outcome of this random event by leaving and rejoining, effectively grabbing a reroll on which random files it receives. If we assume that the host is performing a reroll to increase the likelihood of receiving fake files, we can penalize the host for leaving early a greater amount than the host stands to gain by performing a reroll. We can define 'leaving early' by forcing the host to commit to an exact amount of time of being on the network. Once that time has expired, the host is forced to leave. If the host leaves before that time has expired, the host has left 'early'. The host stands to gain the most from doing a single reroll (multiple rerolls will have diminishing returns), so the network will assume that the host only intends for 1 reroll on average when trying to determine the penalty. We observe that a single reroll means that the host will get a better arrangement of files with approximately half of a standard deviation, meaning if the standard deviation for the number of fake files the host expects to get is 100, then a single reroll means the host can expect to get 50 fake files more than what would be expected without doing any rerolls. The network can assume that no more than 50% of the files are fake. The network can calculate the standard deviation by taking the square root of 50% of the files, times the probability of getting each file, times the inverse of that probability. The penalty for leaving early is then defined by the cost of storing a half standard deviation of files for the amount of time that the host had remaining.

$$Penalty = \frac{1}{2} * \sqrt{(fakes) * \frac{1}{hosts} * (1 - \frac{1}{hosts})} \quad (1)$$

On networks where hosts are expected to have a large number of files, say O(10,000) (this can be achieved by expecting something like 1TB from each host, and then mandating that individual files be at most 100MB - larger files just get split up), the standard deviation is very small compared to the number of files that the host is storing. At 10,000 files per host, the above equation works out to a penalty of 70.6 files. 70.6 is a small fraction of 10,000, and so a host storing an expected 10,000 files will have at most a penalty that is less than 1% of the total expected income for storing the files. This penalty is a reasonable risk for honest hosts to take.

When a host leaves the network, whether early or late, it has the responsibility of transferring all files to the next owner. The host has the ability to decide not to transfer the file, which is the same as corrupting the file. Preventing manipulation upon leaving the network is already covered if the same file corruption penalties are applied to hosts who do not properly transer a file to their successor upon leaving the network.

We want to give hosts a way to deal with file corruption that does not result in them personally paying out for the full refund of the files lost, as this is prohibitively expensive. The purpose of the penalty is not to discourage corruption, but instead to discourage manipulation of file distribution. In the case of wholesale corruption, the goal is either to remove files from a host that has a high volume of non fake files, or to prevent a high volume of non-fake files from being sent to other conspiring hosts on the network, or to prevent a high-volume of fake files from going to non-conspiring hosts on the network. For this case, we need to look at the high probability cases, because a host can predict the outcome of wholesale corruption before executing, meaning that they can choose to execute the corruption only in favorable or unusual cases. We do not care about extremely unusual cases, because a conspiring set of hosts will have very few opportunities to execute on these, and cannot use them to adjust the network by more than a fraction of a percent. We will establish 'extremely unusual' as being 4 standard deviations outside of the expected, which will happen 0.003% of the time. A host moving files favoriably at 4 standard deviations, in the case of 10,000 files per host, will be manipulating 560 files. Therefore we set the penalty of wholesale corruption equivalent to the benefit of moving 560 files, or about 6%, where you pay for the average time remaining on the files. Though a bit scarier than the leaving early penalty, the wholesale corruption penalty is managable.

The distribution of files cannot be disrupted by adding files to the network, as files join with a random seed, are sent to random hosts, and are stored for a fixed amount of time. The distribution of files can be disrupted however by removing hosts when the hosts get an unfavorable random seed, or are able have a set of files sent to a favorable location by leaving, All of the penalities associated with manipulating the distribution of files are expensive, regardless of whether the goal of manipulation involves fake files or not. A conspiring set of hosts comprising 50% of the hosts on the network looking to cause a disparity of 5% between conspiring and non-conspiring hosts would have to move the network in 2.5 standard deviations each direction, when there are 10,000 files per host. Getting the average conspiring host to have 2.5 standard deviations fewer files than what is probabilistically expected would require well over 100 rerolls per host, something that we can assume is economically prohibitive. In conclusion, disrupting the distribution of files per host beyond any reasonable value (defined as a 5% gap) is economically prohibitive.

A full summary of the rules of the network follows:

- Files and hosts join the network with a precise, prepaid lifetime.
- After joining, each file and host is given a random seed.
- Files are placed onto hosts using the random seeds and rendezvouz hashing.
- Hosts are penalized for losing files ('corrupting files') an amount equivalent to how much time the file had remaining on the network, and the uploder is refunded.
- Hosts are penalized for leaving early equivalent to how useful a reroll might be in changing the distribution of fake files on the hosts machine.
- Hosts are penalized for 4 standard deviations of file redistirbution if they lose all of the files - and they are pardoned from any penalities for individual files in that set.

This set of rules enforces the three conditions established at the beginning of the paper, given the assumptions established at the beginning of the paper.

We observe 4 shortcomings with this approach:

- Files need to get new seeds after their original payment expires. This causes turbulence, as the expired file will need to be uploaded to a new location.
- Hosts need to get new seeds after their allotted time expires. This causes turbulence, as the host will need to download a new set of files.
- Hosts can be penalized, and there must be a reliable way to enforce the penalties.
- The number of hashes needed to determine the state of the network is the number of hosts times the number of files. This will not scale.

A better scheme should be looked for.