Low Bandwidth Random File Federating

David Vorick
Nebulous Labs

September 11, 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 wish to use this network to store arbitrary data, however we also wish to use this network as a basis for consensus by proof of storage (a host is allowed to participante in consensus if they are completing a sufficient volume of proof of storage). In protecting consensus, we only wish to make sure that no malicious party can gain control of a majority of the voting power.

Assumptions:

- No party will ever control greater than 50% of the raw storage on the network.
- No party will ever control greater than 50% of the files on the network.
- All non-malicious files are fully compressed and non-redudnat.

The third item is perhaps a stretch.

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

- Every file is stored on a host
- The number of files each host stores follows a normal distirbution.
- A set of hosts controlling less than 50% of the raw storage on the network cannot appear to control greater than 50% of the raw storage on the network.

In proving the third item, I establish that following these two conditions is sufficient:

- A set of hosts less than 50% of the network cannot have a greater than 50% chance of uploading a file to machines they control.
- A set of hosts controlling less than 50% of the files on the network cannot perform any action which results in the expected percentage of files across all machines under their control to exceed 50% of the files under their control.

2 Introduction

This is one piece of a larger federated proof-of-storage cryptosystem that is being built. The system as a whole is far from complete, however the scope of this paper extends only to the method which hosts are distributed files on which they will perform proof of storage. The goal of the larger system as a whole is to provide a cheap and secure way to store files in a decentralized manner. To keep costs as minimal as possible, a few constraints are enforced:

- The amount of expense required to maintain consensus outside the cost of storing files should be minimal.
- The bandwidth consumed by uploading a non-reudundant file should not exceed the size of the file.
- The bandwidth consumed by maintaining the file on the network should be minimal.
- A host should not ever have to release a file unless it is giving the file directly to a new host.
- A host should not ever have to take on a file unless the file is new to the network, or unless the file is being taken from a host that has gone offline. (assuming the file is still recoverable in some way)

Because we expect the hosts to be doing storage proofs on their files, and we expect to use the storage proofs to drive consensus, we must make sure that there is no way for a host to manipulate the files that they are storing such that they can perform proof-of-storage on a greater volume of storage than their economic power allows. This is achieved by using a special type of addressing to assign files to hosts.

Rough Draft note: I'm wondering if some of the things that I am doing here are in some way convoluted or unnecessary, and I'm looking for security flaws, proof shortcomings, as well as suggestions on improving/simplifying the system.

3 Addressing Scheme

There is a 80bit addressing space which helps to determine which hosts will store which files, containing

$$2^80$$

slots for files. Collisions are permissable - multiple files or hosts can be in the same slot. This address space is only partially in use at a time, depending on the number of hosts in the network. For each host, $2^24 slotsareopenonthenetwork.Eachhosthasa'spanning$

$eachslotonthenetworkwillprobabilisticallybecoveredby$

When a file is added to the network, it is assigned a random seed 256bits in length. The seed is assigend from an external source of entropy - it is not the hash of the file nor has any relationship to the file. This seed is truncated to produce a random slot in the 80 bit address space. If the slot assigned to the file is outside of the active space, the hash of the seed is taken and used to produce a new slot between the 0th slot and the initial slot. This process is repeated until the file ends in a slot within the active address space. **Example:** We have a full address space of 512 slots with an active space of 100 slots. A new file is randomly assigned a seed that is used to produce a random slot between the 0th slot The seed is hashed to prodouce a new value, and this value is used to choose a new slot between index 0 and index 386 (inclusive). The new slot is 105, still outside of the address space. This process is repeated until a slot value appears that is between 0 and 100 inclusive. The expected number of hashes is approximately equivalent to the log of the number of slots that are inactive minus the log of the slots that are active.

When a file is added to the network, a slot is chosen within the active space. This slot is likely to be covered by $2^6 potentialhosts, yetonly 1 hostisallowedtostorethefile.Thisconfl$ $explainedlater.$

When a host is added to the network, the active addresses are expanded by $2^24 slots.Thehostisthengivenarandomseedderivedfromanexterna$

As a host is added to the network, new address space is opened up. We wish to put files in this new address space at random, taken from the other slots on the network. This is achieved by looking at the iterated hashes used to find a file's current location. If a slot in a previous iteration opens up, that slot is given priority for the file, and so the file is moved from its current address to the now-available space.

When a host leaves the network, every file that the host was storing is given a completely new seed.

4 Byzantine Attacks

$of 2^30 slots, centeredaroundthehost's slot.Thismeans$ We don't care what goals byzantine attackers have,

$2^6 hosts.Ahost's$ so long as the following properties are maintained: $featureofthene$

- Every file is stored on a host.
- The number of files each host is storing follows a normal distribution within a provable bound.
- A host controlling less than 50% of files cannot manipulate the network such that it will probabilistically be able to store greater than 50% of files on itself.
- A malicious body of hosts less than 50% of the network in size cannot manipulate the network such that a file being uploaded will have greater than 50% chance of landing on a member of the malicicous body.