Membrane - Distributed File Backup - Literature Review

Dominic Hauton

January 9, 2017

Abstract

An application in which users can join a large public network of untrustworthy intelligent storage agents to trade their available file space. This will allow the user to store files safely on hostile computers, increasing availability and redundancy. The hosts will act as intelligent agents, and trade space when they communicate with one another.

0.1 Introduction

Distributed storage is a well studied and explored domain with clear advantages over bare metal. In order to implement a peer-to-peer distributed agent based storage system we must first decompose the problem into several parts and explore the advances made in those fields.

With the advent of cheap high-speed internet users are now able to use monolithic cloud services to backup data. These tend to be expensive for anything but small amounts of storage and many people have expressed security concerns over holding their data in a data centre owned by another company. Especially if their data leaves the users country where data protection laws may be different.

Over time personal storage capacities for users have also increased. It is now common for computers to come with large amounts of hard drive space which is often not filled to capacity. The project proposed promises to swap this free hard drive space to back up other users' data in exchange for their surplus space to backup your data.

To simplify the process, the system will be able to negotiate contracts of varying complexity for space allocation on another machine, in exchange for space on itself. Unlike most distributed databases and cloud storage solutions frequent down-time will be expected across devices and contracts will have to account for this.

The challenges of such a system include a way of only copying parts of a file during updates to reduce bandwidth usage. An effective way of managing where file chunks are stored. A way to encrypt the files on the remote host and a way to authenticate that the host still holds the file.

Part of the required research also includes exploring the more recent technology of intelligent agents, in which we are most interested in trust metrics. Intelligent agents will allow the software to make storage decisions based on the trust and reputation of other nodes in the network.

0.2 History of Problem

A simple file backup can be imagined as simply copying a file to another location. In order to keep the duplicated file in sync they must be compared. The program diff solves this through finding the longest common subsequence between files. In order to improve performance hashing, presorting into equivalence classes, merging by binary search, and dynamic storage allocation are used. [Hunt and MacIlroy, 1976]. This allows the user to view changes and copy the file over again if required.

0.2.1 Rsync

In a networked scenario, bandwidth from source to destination is at a premium. Rsync, introduced in 1996 presents a much better solution. Instead of copying the entire file multiple times, the changed file chunks can be copied instead. [Tridgell et al., 1996]. Rsync splits the file into chunks and calculates a weak rolling checksum and strong MD4 checksum for each block. Unless there is a match, the weak checksum moves across the file to find a possible offset for matching data and confirms the offset with the strong checksum. The gap can then be requested to be copied. This results in a lot less data being copied than there would be with a diff file. [Tridgell et al., 1996] This combination of weak and strong checksums has been used across multiple distributed systems including low-bandwidth file systems [Muthitacharoen et al., 2001] and high performance transactional object stores. [Stephen et al., 2000].

Multiround Rsync improves on the rsync algorithm by allowing for more communication to lower bandwidth. Blocks of smaller and smaller sized are used to find holes in the old file in each round and the file until the minimum block size is reached and a copy occurs. [Langford, 2001] This works better than standard rsync in situations where the source file has been changed in many places distributed around the file.

Rsync requires both old and new copies of a file to exist on the host system during an update. This issue has been addressed by creating an in-place rsync (ip-rsync) that uses techniques used in delta compression and Lempel-Ziv compression to move the areas of the file around. In ip-rsync file sender sends add and move commands to the destination in an order that guarantees no files will be overwritten. [Rasch and Burns, 2003]

0.2.2 Git

Git is an improvement on Rsync as it provides both version history and minimises data transfer. To keep storage simple, a copy of the whole file is stored and a reference is put into the version history. By storing old files locally operations are fast. This is also an important distinction from other version control systems, because it means the systems can operate without a centralised server. To reduce file duplication all files are referenced using their SHA-1 hash. This means you can be sure the contents of the file hasn't changed since you last read it.

Git also uses a 3 stage workflow. A working directory, where the current files are stored, a staging area and a .git directory. The staging area prepares your next commit and then it is finally committed. When the staging is complete the change is irreversibly stored. This is a good approach that will be adopted in the final software solution. It will allow incrementally finding changed files, and assessing the need for a new version number to be saved.

0.2.3 Bittorrent

The BitTorrent protocol is a mechanism for sharing files between swarms of hosts. As BitTorrent splits files into parts, users start sharing data even before they have received the full file. Each file has a SHA-1 identifier, similar to Git. [Qiu and Srikant, 2004]

If a user wishes to download a file from the swarm, the user downloads a metadata file from the web and locates users sharing the data using a Tracker Server, Distributed Hash Table (DHT) or Peer Exchange (PEX). [Cohen, 2008]

A Tracker server is a centralised store of all current connected users along with how much of the file they hold. This approach is vulnerable to exploitation by authorities as all of the data about a swarm is strored on a single server and as a result cannot be used for the proposed system.

A DHT contacts other known users for information instead of a centralised server. The Mainline DHT as outlined in BEP No.5 is based on the Kademlia protocol that allows for decentralised peer discovery for a particular piece of content.

PEX is a method for two clients to share a subset of their peer lists. Coupled with DHT, PEX removes a vulnerability from the Bittorrent network by allowing fully distributed bootstrapping, tracking and peer discovery.

A DHT with a form of PEX a tried and tested way of successfully mapping and finding files on a network and will be used within the proposed project.

0.2.4 Resilio

Resilio Sync is an example of a distributed file storage system that utilises the BitTorrent protocol to automatically synchronise folders between a user's systems. It is not a cloud backup solution and not intended as a form of off-site storage. There is no distributed file system and as a result, no redundant data block algorithm adding complexity. [Farina et al., 2014]

As Resilio Sync uses DHT to transfer data, there is no central authority to manage authentication or log data access attempts. This makes it difficult to determine whether a file has been accessed by another user. [Farina et al., 2014] As a result in the project the assumption will be made that everyone in the network has access to all encrypted file chunks. To access and reassemble a file, a user will be required to request all of the file chunks individually and then locally reassemble them.

0.2.5 Storij

Storj is a peer-to-peer cloud storage network which aims to allow users to store files on a decentralised platform. The platform takes special care to provide protection against Sybil attacks and other forms of fraud. [Wilkinson et al., 2014]. To store files it stores encrypted hashed shards on the network. In order to provide proof of storage it uses Merkle Audits and pre-generated audits with hash-challenges to determine whether the client still holds the required data. By adding a seed to the hash-calculation the client can enforce the workers are still in possession of the data. It prevents the client cheating a farmer through using blockchain proof-of-existence to keep both parties honest.

The most efficient form for proof of storage is through using a deterministic heartbeat. Using Feistel permutations data can be verified with $n + 2\sqrt{n}$ challenges. Erasure encoding is added to shards to detect any minor changes to the data. This is less I/O intensive than a full heartbeat, but still allows an attacker to complete heartbeats with only a data integrity of 1/n, where n is the number of nodes holding the data.

In order to add extra protection to files, we can use erasure encoding to allow file recovery if one of our shard types is lost. This can be investigated in our software, however, as the shards are expected to change on a regular basis because of versioning, this may not be possible.

To prevent Sybil based attacks, Storij encrypts each shard segment with a different salt. This stops workers completing proof of storage on behalf of another node.

0.3 Peer Admission

The first step in designing the distributed file system is locating other peers within the swarm. This is accomplished through Peer Admission. Once the first peer is found data within the swarm can be located using a DHT which guarantees content can always be found.

0.3.1 Bootstrapping

There are two types of PTP networks which must be examined:

- *Asynchronous*
- *Synchronous*

Within Synchronous networks the number of nodes on the network is constant and all of the nodes are aware of each others existence. This does not allow storage networks to scale but it does allow data to be kept private. [Saxena et al., 2003] This is the simplest and first approach that will be taken in locating nodes within Membrane.

In most current P2P systems such as Gnutella [Klingberg and Manfredi, 2002], Chord and Tapestry as well cryptocurrencies such as in Bitcoin and Litecoin a bootstrapping node is contacted, which provides information about what clients are currently online. Once a bootstrapping node allows the client to find the edge of the swarm, more information can be found using peer exchange.

Within a local network we can also use Universal Plug and Play to find other nodes within the local network. This prevents an external call to a bootstrapping node and as a result is less prone to attack.

Within Membrane, I plan to use a combination of Asynchronous and Synchronous techniques. Users will try to bootstrap from their last known neighbour nodes on the network. Only if this fails, and with the user's permission will they contact a centralised bootstrapping node.

A further extension of this, would be to allow hosts to provide a DNS name along with their IP. Users that have setup Dynamic DNS (DDNS) [Bound and Rekhter, 1997] would be able to locate each other without the help of a bootstrapping server.

0.3.2 Peer Exchange

When bootstrapping is complete new Peers can increase their knowledge of the network through Peer Exchange. This is used by Bittorrent to help share swarm information with other nodes

0.4 Data Allocation on External Nodes

0.5 Distributed File System

0.6 Authentication

0.7 Untrustworthy Hosts & Encryption

0.8 Conclusion

Bibliography

Bound, J. and Rekhter, Y. [1997], 'Dynamic updates in the domain name system (dns update)'.

Cohen, B. [2008], 'The bittorrent protocol specification'.

Farina, J., Scanlon, M. and Kechadi, M.-T. [2014], 'Bittorrent sync: First impressions and digital forensic implications', *Digital Investigation* **11**, S77–S86.

Hunt, J. W. and MacIlroy, M. [1976], *An algorithm for differential file comparison*, Citeseer.

Klingberg, T. and Manfredi, R. [2002], 'The gnutella protocol specification v0. 6', *Technical specification of the Protocol* .

Langford, J. [2001], 'Multiround rsync'.

Muthitacharoen, A., Chen, B. and Mazieres, D. [2001], A low-bandwidth network file system, *in* 'ACM SIGOPS Operating Systems Review', Vol. 35, ACM, pp. 174–187.

Qiu, D. and Srikant, R. [2004], Modeling and performance analysis of bittorrent-like peer-to-peer networks, *in* 'ACM SIGCOMM computer communication review', Vol. 34, ACM, pp. 367–378.

Rasch, D. and Burns, R. C. [2003], In-place rsync: File synchronization for mobile and wireless devices., *in* 'USENIX Annual Technical Conference, FREENIX Track', Vol. 100.

Saxena, N., Tsudik, G. and Yi, J. H. [2003], Admission control in peer-to-peer: design and performance evaluation, *in* 'Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks', ACM, pp. 104–113.

Stephen, Z. H., Blackburn, S. M., Kirby, L. and Zigman, J. [2000], Platypus: Design and implementation of a flexible high performance object store., *in* 'In Proceedings of the Ninth International Workshop on Persistent Object Systems', Citeseer.

Tridgell, A., Mackerras, P. et al. [1996], 'The rsync algorithm'.

Wilkinson, S., Boshevski, T., Brandoff, J. and Buterin, V. [2014], 'Storj a peer-to-peer cloud storage network'.