CoStor, a distributed backup solution

Robert Phipps

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2020

Abstract

CoStor is a proof of concept cloud backup solution, making use of simple to manage HTTPS requests for all communication and synchronisation across the system, allowing for easy integration into existing environments. Authentication is managed by tokens and role-based authorisation, with configuration for the client set with an easily deployable YAML file.

This implementation makes use of two primary components, a command line client application, which builds a database of and fingerprints the directory tree to be backed up, and a Django based server application, which is used as the sync endpoint for off-site storage of backup data. These communicate using the HTTP API to identify file binaries and metadata that need to be transferred, and a multi-stage, resumable upload process is used to upload large files to the system.

It uses generic objects and datastructure fingerprints to allow server-side deduplication of backed up data across clients, and allows backup ”snapshots” to be restored as archives from the web UI of the server application.

Acknowledgements

This project makes extensive use of concepts and architectures explored in this paper from Paul Anderson and Le Zhang: Fast and secure laptop backups with encrypted de-duplication [1]

Many thanks to Paul Anderson for his supervision of this project.

Table of Contents

Chapter 1

Solution overview

CoStor is designed as a turnkey solution to the problem of maintaining reliable system backups within an SME with multiple sites, such as a confederation of schools. Instead of using expensive and bandwidth intensive cloud storage services for offsite backup, CoStor is designed to hold a complete local backup on-site within the CoStor server, as well as automatically replicating backup data across a group of federated instances of the server software, ensuring that there is always at least two redundant copies of the backup datastore in two different physical locations.

To simplify networking requirements for deployment, all communication between clients and servers, both locally and between sites, makes use of standard HTTPS requests. This negates the need for complex multi-site VPNs, and simply requires a single TCP port to be forwarded to the server from the internet.

The backup datastore's metadata and directory structures are maintained inside an SQL database and can only be modified over the HTTP API, reducing attack surface compared to making use of more traditional file transfer methods like FTP, NFS or SSHFS.

All management is completed through a simple web UI, where agents can be added, authentication tokens can be created and the backup history subsequently monitored. Users can browse the directory trees for each backup ”snapshot” in the event that a file needs to be recovered. A full backup restore can be completed by requesting a restoration archive, whereby the server will build a complete archive of a snapshot, pulling data from its local datastore, or from other sites in the event of a remote restore[1].

This system makes use of Django for server-side components, a Python web framework with fantastic ORM and enforcement of best-practices. The client side database is managed with PonyORM, running on top of SQLite.

[1]Remote restore has not been fully implemented

1.1 Goals of CoStor

Given the target organisations for CoStor, there are some specific goals that need to be targeted during development.

- **Reliable backups**

  As should be very obvious, being a backup solution, CoStor needs to be able to reliably manage and maintain backups for a network. This includes protections such as an ”append-only” API for backup clients, validation of uploaded data, and mitigations against the most common reasons a backup may be called upon such as accidental deletion, user errors, hardware failure and ransomware style attacks.

- **Simple restores**

  As this system is targeted at small organisations which may not have their own full-time IT support staff, restoring from backups should be straightforward for an end-user. This is achieved through the use of a self-service web UI.

- **Robust security and audit logs**

  Backups almost always contain confidential information, so CoStor needs to be able to manage permissions on a granular user-by-user basis. It also needs to include audit logging for all operations on the system. Any offsite storage and ”data in flight” needs to be strongly encrypted to protect confidentiality.

- **Low maintenance**

  Systems tend to be forgotten about, and in the case of backups, often you only notice something hasn't been working once you need to restore something[2], so CoStor needs to automatically maintain its database and include robust database integrity checking to ensure that the system is ready when the user needs it most.

- **Simple deployment**

  Again, CoStor needs to be deployable by inexperienced IT support staff without prior knowledge of network filesystems, command line interfaces or web development. This can be achieved by making use of clever packaging and deployment strategies such as Docker for server components and zero-touch installation scripts for backup clients. By making use of standard and well understood protocols such as HTTP(S) for communication between components, compatibility with most network architectures should be maintained, without the requirement for complex network share configurations, multi-site VPNs and authentication systems.

- **Centralised management and monitoring**

[2]GitLab found this to their cost in 2017 after discovering their backups hadn't been running for some time: `https://techcrunch.com/2017/02/01/gitlab-suffers-major-backup-failure-after-data-deletion-incident/`

As this is designed to be deployed over a large number of client PCs, ensuring configuration is correct could be challenging. As such, CoStor will allow setup to be pushed to clients using a simple config file. Backup logs will also be maintained on the server so that an administrator can monitor their entire estate from a single place.

- **Distributed and fault-tolerant file stores**

  Leaving the best to last, CoStor's standout feature will be that backups can be automatically replicated between federated instances of the CoStor server, over a zero-configuration HTTPS link. The system should be able to recover from the loss of a server without any data loss, and allow restoration of data originating from any site from any of the remaining instances of CoStor within the network.

  (Unfortunately, distributed backup was not able to be implemented ahead of the deadline for this project.)

1.2 Limitation of scope for the purposes of this project

This project was proposed as a two-year task, as part of the MInf course. Due to circumstance changes, this project has been condensed into a single year as part of the BSci. This has required many of the user interface elements and advanced features to be removed from scope.

As such, in its current state, CoStor is a single-server solution, which would require a traditional second tier of redundancy for the backup store, such as scheduled `rsync` tasks to a remote filestore, or hardware redundant storage including LTO tape archives. Support for these is not included in the CoStor proof of concept provided here. The methodology and structure of a potential cross-site replication implementation is however included in this report, as it offers context to a number of design decisions. As data currently does not leave the corporate network, backups are not encrypted at rest, and only with standard HTTP TLS in flight. There is no need to restrict access to specific systems' backups as the CoStor administrator will already have access to all users' documents as a network administrator.

The management of local filesystem metadata was considerably more challenging than the timescale for this project anticipated, with the CoStor client requiring far more logic within it than was intended. This resulted in available time for the server application to be limited.

1.3 Exploration of existing solutions

There are many packages in existence which incorporate a subset of the features targeted by this project, however most either focus on the synchronisation features with some limited support for file history, and no robust backup capability, whereas others rely on cloud storage infrastructure as the backend, distributed datastore which either necessitates the use of a commercial provider such as AWS S3[3] or BackBlaze B2. Both of these greatly increase cost and introduce a reliance on a third party.

1.3.1 Overviews

A number of existing products have been selected as they offer the closest functionality to that targeted by CoStor. Here we take a high-level look at featuresets and methodologies used by these solutions:

1.3.1.1 Syncthing

Syncthing is a service targeted at consumers who want a self-hosted alternative to commercial cloud storage and synchronisation services such as Dropbox, Google Drive and OneDrive. The agent software can be configured to sync any file changes between two or more devices, allowing for access anywhere, with a form of georeplication. It also requires fairly minimal network configuration, just needing a pair of ports to be opened on at least one of the nodes to allow discovery of other agents.

A very large distinction has to be made in the fact that ”Syncthing is a continuous file synchronization program”[2], in that it isn’t designed to create restorable snapshots of your data, and therefore is completely unsuitable for robust *backup* of important information.

More information is available at `https://syncthing.net` [2]

1.3.1.2 UrBackup

UrBackup is closer to CoStor in its goals, as is specifically built to be used as a backup system. It can auto-discover agents on the network, and begin incremental backups to its server software. It also supports full image backups of NTFS formatted drives with bare metal restore. UrBackup has many of the features targeted by CoStor, including a simple web interface for management, however it does not include any built-in support for georeplication.

More information is available at `https://www.urbackup.org/index.html` [3]

1.3.1.3 Hermes

Hermes is an ”open-source redundant distributed storage network”[4]. Although it doesn’t come pre-packaged with components allowing it to be used as a turnkey backup system, it is worth exploring as it does specifically target the geo-replication features

[3] Amazon Web Services’ bucket storage solution

for the purposes of backup that CoStor is looking to integrate. It promises fast, encrypted and seamless replication and sharding of data across nodes in the network, making use of LZMA compression to increase performance.

This would appear to be a very promising option to integrate as the backend storage for CoStor, however it looks like the project is very much stale, with the last commits being made in late 2014. As such, its codebase is somewhat limited in utility, given its lack of ongoing maintenance. It is also written in Go, with limited documentation, which is not a language that I am familiar enough with to begin work on reviving.

More information is available at `https://github.com/Hermes/hermes` [4]

1.3.1.4 Bacula

Bacula is an open-source and very mature backup framework, with tools to allow a multitude of network configurations. Unfortunately its flexibilty does result in the software being complex to configure. CoStor is targeting small organisations with limited in-house IT support capacity, so this would likely be too complex to deploy without the assistance of external contractors. Bacula is also available in an "enterprise" edition[**?**], which includes support as part of the subscription cost, however this version is both closed-source and not inconsiderably expensive.

More information is available at `https://www.bacula.org` [5]

1.3.1.5 Amanda

Amanda is another backup-specific solution, again with a "community edition" being accompanied by a commercially supported "enterprise" version of the software. Like the other systems investigated, it targets backup to local storage, NAS drives and traditional tape backup libraries, so would not be suitable for the use case targeted by CoStor.

More information is available at `https://www.zmanda.com/amanda-community-edition.html` [**?**]

1.3.1.6 GlusterFS

The Gluster filesystem is a "software scalable network filesystem"[6], which allows systems administrators to build complex filesystems across multiple physical machines using commodity hardware. Gluster is targeted at engineers designing private cloud environments which need large volumes of storage accessible from many devices, and has a huge number of configuration options to allow, among other things, redundant replicated storage across machines. It also has support for filesystem filters that can offer encryption at a filesystem level before data is transmitted between nodes.

Unfortunately for CoStor's usecase, this system is built on the assumption that the Gluster volumes are configured at the point of deployment, by the same user on all sites. It also is primarily managed through a command line interface, with limited Python bindings, and is targeted at an architecture where all nodes have rights to see data within the volume. Finally, it is based around a master-slave architecture, which is unsuitable if CoStor were to be using a shared volume for all nodes. It would be possible to wrangle a Gluster configuration to match what is required by CoStor, where each site has its own volume, with its own georeplication session, however this would be very difficult to automatically provision from the CoStor web interface, and should be mostly unnecessary given the data structure used for backups.

More information is available at `https://www.gluster.org` [6]

1.3.2 Case study: Git

Git is one of the most well known versioning and distributed storage systems, and works in a very similar way to how CoStor is intended to behave. It maintains a local database of all file data, storing file history within the records for each file. It supports delta-replication to one or many local or offsite Git servers, with the ability to recover data from any point on time that has been ”committed” to the history.

1.3.2.1 Data storage

The Pro Git Second Edition documentation [7] states that ”Git is a content-addressable filesystem. [...] What this means that at the core of Git is a simple key-value data store.”

Everything in Git is stored as generic objects, with their SHA-1 hash making up their unique ID within the datastore. These can be data objects, tree objects and commit objects, all interlinked by ID to form a representation of the versioned filesystem. These objects are stored as files within the `.git/` subdirectory created when Git is initialised. (Figure 1.1)

All directory structure and file metadata is stored in the tree object for a commit, with file data being stored in a separate ’blob’ object and referenced by the tree. This allows a single copy of the file to be used as the ”backup” copy for multiple commits without requiring multiple copies of the file to be stored, reducing disk usage considerably as most commits only touch a few of the files within the repository.

Git also selectively processes file metadata, for example only including the three UNIX file modes for normal files, symlinks and executable files. This simplifies processing and makes behaviour predictable for the end user. It also mitigates any chance of accidentally interacting with special devices such as serial ports and pipes, which could confuse the system.

Occasionally, the client creates delta compressed ”packfiles” of the objects, explained later.

1.3.2.2 Data transfer

For data transfer, Git can use either an SSH based protocol, which they call ”The Smart Protocol”, or a ”Dumb” HTTP based protocol. For the purposes of CoStor, this HTTP

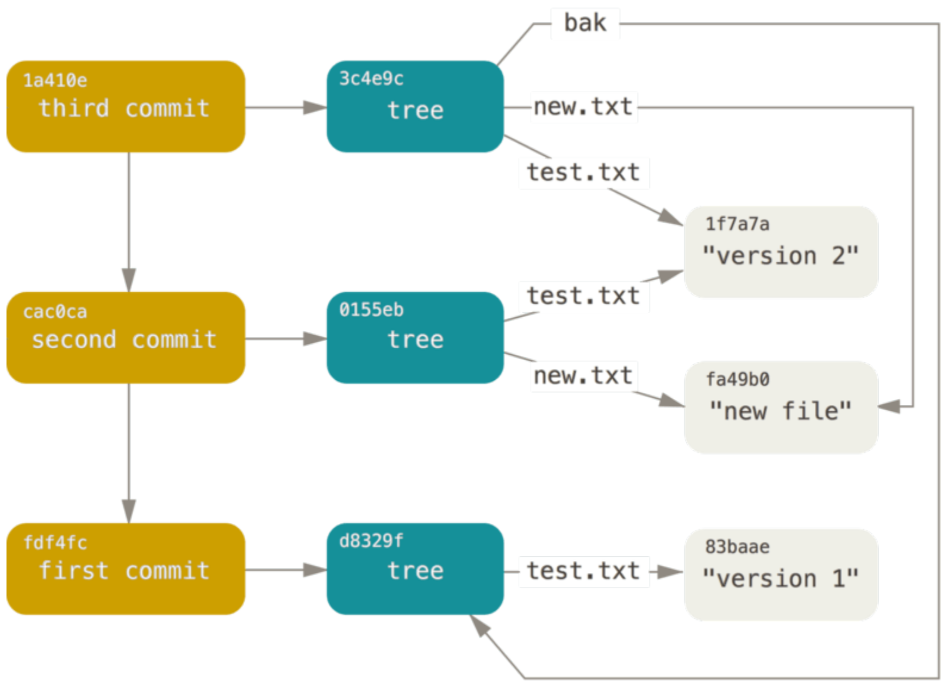


Figure 1.1: Representation of the Git content addressable filesystem [7]

based protocol is of more relevance, as we are trying to avoid reliance on ”exotic”[4] protocols.

First, the client makes a GET request to pull down the SHA-1 IDs of the branch that the user is interested in, or in Git terminology, the client requests the HEAD reference. Once it has that reference, it is able to make a further request to the specific commit required, by its SHA-1 ID. The commit contains a reference to its tree object, and the parent commit, which gives the client everything it needs to rebuild the entire commit history. Some data can be bundled into packfiles, in which case the Git server will simply HTTP 404 and the client will have to make a few more requests to work out the packfile containing the commit.

For making commits, the client first builds packfiles, which is where it takes all of the objects and performs delta compression to reduce the transfer size. It looks for files that are named and sized similarly before trying to calculate the difference between the files. This can drastically save on space compared to uploading the ”loose” data objects. And, as these packfiles contain an entire copy of the Git datastore, this can be uploaded as one file to the server.

1.3.2.3 Evaluation

This is a very elegant solution to the problem, and allows Git to be a very simple and portable version control system. However, its reliance on a standard filesystem for the datastore does potentially make management of that data more complex, as there is no

[4]Yes, SSH isn’t exactly exotic, but port 22 is commonly blocked on company firewalls, along with networks that use an HTTP proxy for web filtering and monitoring

support for foreign key relationships or the ability to make use of an ORM to ensure the database integrity is maintained.

In the case of Git, there is normally at least one other copy of the database, so if it is corrupted on the local machine, it can be rebuilt from that copy.

The generalised data structure of generic ”objects”, representing everything the system needs to keep track of, however could be very easily replicated within an ORM. Storing the binary data separately from the file metadata can also enable us to deduplicate files across systems, with not only multiple trees from the same repository sharing a blob database, but multiple trees from multiple systems running the backup client software and sharing the same server instance.

An important thing to note, is that there appears to be no use of filesystem snapshots: Git relies on files not being written to or changed while it indexes the repository. This means it would not be suitable for backing up live application datastores or constantly changing data such as web server databases unless the database server application is shut down while the commit is made.

1.4 Deployment topology

The basic topology of a CoStor network would be as follows:

- **Clients** (many):

  The CoStor client software is installed on any systems which are to be backed up. It communicates over the local network to the site’s local instance of the CoStor server.

- **Servers** (one per site/network):

  There should be one instance of CoStor server on the internal private network of any site that has clients to be backed up. There could be multiple instances running in one local network space, but they would operate as separate ”sites” within the software. The servers are the primary file store for a site, and manage replication between federated instances on other sites.

- **Federated servers** ($>$ 2 across multiple locations):

  These are additional instances of the server application, and can be on separate networks and in different physical locations. These communicate over the internet through a replication API to keep a redundant copy of all data from all servers. They can restore data from any site within the group in the event that a site’s server fails, assuming the target site’s encryption key is provided. The replication is explained in more detail in Chapter 4.

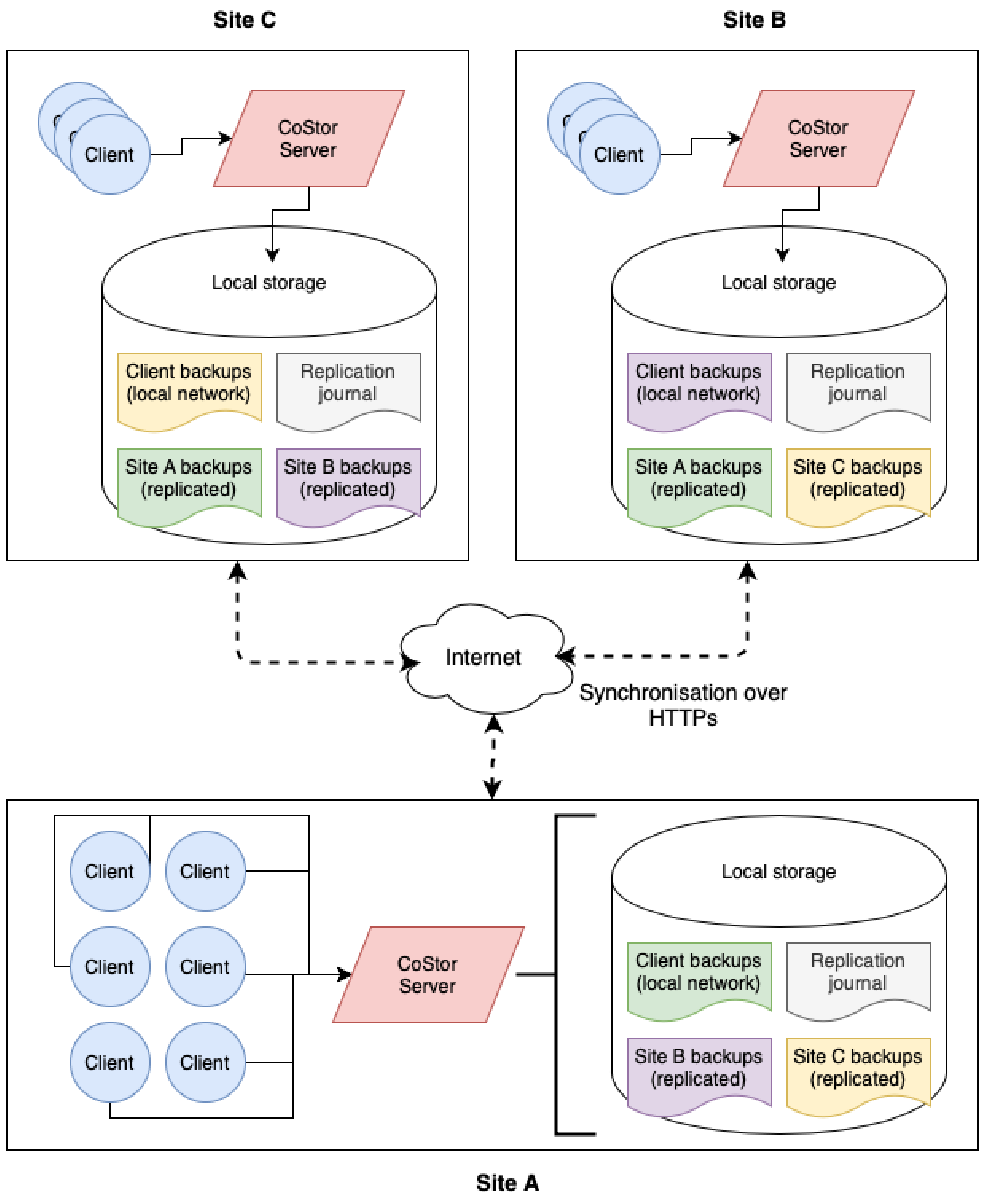

Site C
Client
CoStor Server
Local storage
Client backups (local network)
Replication journal
Site A backups (replicated)
Site B backups (replicated)
Site B
Client
CoStor Server
Local storage
Client backups (local network)
Replication journal
Site A backups (replicated)
Site C backups (replicated)
Internet
Synchronisation over HTTPs
Client
Client
Client
Client
Client
Client
CoStor Server
Local storage
Client backups (local network)
Replication journal
Site B backups (replicated)
Site C backups (replicated)
Site A

Figure 1.2: Example of a multi-site deployment of CoStor

1.5 Specimen use case

The initial inspiration for this project was a use case within a confederation of small, state controlled schools. Each school currently spends a large amount of money on a third-party managed cloud backup solution - in the region of £500 per year for only 20GB of backed up data. This meant that they were unable to backup considerable portions of their data, leaving some of their media based teaching resources vulnerable.

The purpose of the confederation is to allow the members to share resources and purchasing power for resources that can be beneficial to all of them.

CoStor could be utilised by these schools to allow them to geo-replicate their backups among themselves, on commodity server hardware, allowing them to no longer be reliant on expensive cloud backup solutions. Each site would have a single CoStor server appliance with their file servers, databases, Microsoft Active Directory databases and any other critical data being backed up by instances of CoStor client, allowing backups to complete rapidly across the high speed LAN, with replication to other schools' CoStor servers being carried out overnight when the internet connection is quiet.

The system should be as close to turnkey and zero-maintenance as possible, allowing it to be set up by their IT support contractors, and left to work for years at a time without much interference.

1.5.1 Revised use case

To accommodate for the limits that have been applied to the scope of the project, the current implementation is based off the following use case:

CoStor could be used by a small organisation, which requires a central backup for specific directory sub-trees on networked workstations. The CoStor client is deployed along with a config file to these systems, and scheduled with `cron` to run out of hours, when the systems are not in use. These back up to the shared CoStor server, which deduplicates the binary data of stored files to reduce disk usage when identical files are backed up from multiple systems.

The server application would be managed through a web UI by the server administrator, and any backup archives can be downloaded through this interface by them for any agents they manage.

Chapter 2

Client implementation

2.1 Summary

CoStor clients are designed to be managed centrally through the CoStor server web interface, therefore they require no GUI of their own, just a way to set the target server instance, required directory tree to backup (the backup root) and their authentication token. These are set using a `.YAML` file, which is easy to push to clients with standard network management tools such as Microsoft Group Policy and Logon Scripts.

The client is built around Python 3, and PonyORM working in parallel with an SQlite database. This means there are very few dependencies, once the Python project is bundled, and allows cross-platform development with a shared codebase.

The current client has been developed and tested on MacOS version 10.15 Catalina with an APFS filesystem. Porting file metadata processing to other filesystems is a non-trivial task, due to differences in the representations of this information. APFS is very similar to UNIX filesystems such as ext4, so should be compatible, however this is untested.

2.2 The backup process

The backup process works as follows:

1. **Build directory structure hashes**: The client uses the `HashTreeMaker` class to build a simplified version of the directory tree being backed up, traversing through the tree bottom-up, and building objects for every item it encounters, each item identified by a hash based off of its attributes and the hashes of all its children. File data is also hashed and stored with a path to the file, to be used for backup later.

2. **Local metadata database update**: The client keeps a local database of all backup ”snapshots” as trees of objects, created by the hasher. It checks for an existing snapshot, and uses this to identify any subtrees within the backup root

which have changed since the last backup run. It then writes any changes to the database as part of a new snapshot.

3. **Authenticate with server**: A quick test is made to ensure that the client's credentials are valid and that the server is accessible.
4. **Create snapshot definitions**: Entries are created on the server to represent the backup "snaphot" within the database, and assigned to the agent.
5. **Push changed metadata objects**: The server is queried to see if it has seen any objects with matching IDs (hashes), and any objects that are missing are pushed as a single JSON object. The IDs of unchanged objects that are used in this new snapshot are also pushed so they can be attached to the snapshot.
6. **File "primes" are pushed to the server**: Again, the server is queried to check for existing copies of any of the files that are included in this backup snapshot, by hash. For any files that are missing, the client uses the path included with the object to find a copy of that file, and pushes it to the server, in multiple chunks.

2.3 Large file upload API

To allow large files to be uploaded over standard HTTPS connections, with the ability to recover from connection issues, files are split into (currently 100MB) chunks, which are individually hashed, and then pushed in sequence to the server. This process is detailed in Figure 2.1.

Integrity is ensured by having the client create an "upload session" before pushing chunks of files. This session includes the file hash and the number of expected parts. Each part is also uploaded along with its hash and sequence number, which is verified before the data is appended to the file in the server's filesystem. Once the final part has been received, the server checks the completed file against the hash given in the upload session before marking the file as complete and closing the session.

Chunks can be uploaded over any period of time, and chunks for one file/session can be uploaded interspersed with chunks from other files, the only constraint is that chunks for a session are received in correct sequence order.

Should a chunk fail its upload, due to a verification error or network instability, the client is able to retry this chunk once network has been restored, allowing uploads to be "resumed", at least to the resolution of the chunk size.

2.4 SQL database and other backup plugins

To facilitate safe and reliable backup of specialist server applications, it is necessary to perform steps to either shut down or pause its processes before working with its underlying datastore. These would be implemented as plug-in modules to the client, which could be configured along with other backup parameters.

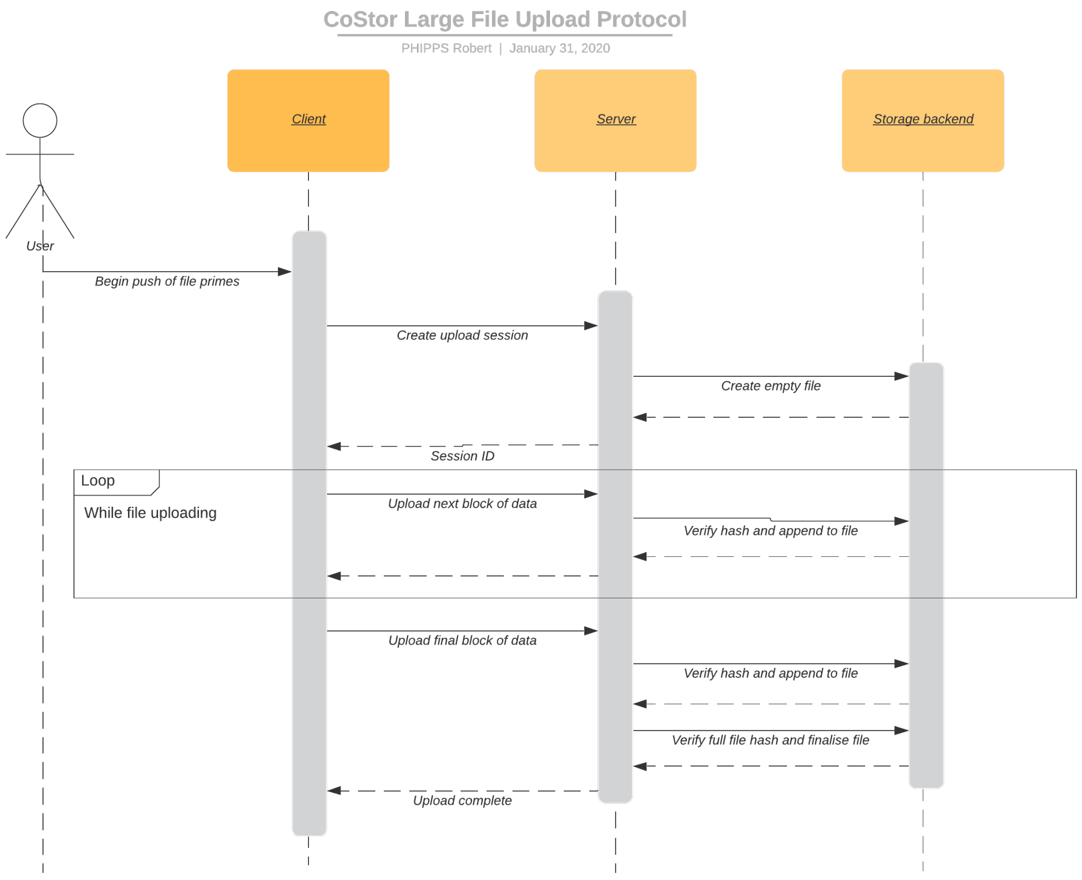

CoStor Large File Upload Protocol
PHIPPS Robert | January 31, 2020
Client
Server
Storage backend
User
Begin push of file primes
Create upload session
Create empty file
Session ID
Loop
While file uploading
Upload next block of data
Verify hash and append to file
Upload final block of data
Verify hash and append to file
Verify full file hash and finalise file
Upload complete

Figure 2.1: UML sequence diagram for large file upload

Taking the example of Microsoft SQL Server, a database dump can be created with the following short SQL command that could be easily wrapped within in a PowerShell script:

```
USE SQLTestDB;
GO
BACKUP DATABASE SQLTestDB
TO DISK = 'c:\tmp\SQLTestDB.bak'
           WITH FORMAT,
              MEDIANAME = 'SQLServerBackups',
              NAME = 'Full Backup of SQLTestDB';
GO
```

Figure 2.2: An example SQL script to create a backup from a running instance of the Microsoft SQL Server, source: Microsoft Docs[8]

The below applications were investigated as candidates for backup plugin development:

- Microsoft SQL Server
- MySQL server (Windows/Linux)
- Microsoft Active Directory Server[1]

Unfortunately, the current implementation of CoStor client does not include support for such plugins, although integration should be trivial.

2.5 Filesystem snapshots and integrity protection

If the user intends to backup rapidly changing data which is always being accessed by another process (such as a operating system core files), it is necessary to ensure that files cannot change during the backup process. One way of achieving this is to create some form of snapshot of the filesystem while the backup is running. This is currently being investigated and will likely make use of LVM, Microsoft Shadow Copies and Time Machine on Linux, Windows and MacOS respectively, however is non-trivial to implement given the differing systems available across both the operating systems and filesystem types.

The current implementation assumes the subtree being backed up is not going to change during the backup process.

[1]https://docs.microsoft.com/en-us/windows/win32/ad/backing-up-an-active-directory-server

2.6 The HashTreeMaker

This class is the engine behind the creation of the filesystem metadata snapshot, and builds simple objects that can be stored in the relational databases used by CoStor. As part of the object creation, metadata is pulled from the `os.stat` command, and unique ids are generated based off SHA-1 hashes of both the metadata, and, in the case of files, binary data.

An early standalone implementation of the HashTreeMaker used during development to dump directory structure to JSON for debugging is located in the `costor_hasher` directory, however the version used within `costor_client` has been through many more iterations as it was integrated with the rest of this application.

All database objects are managed by PonyORM[2] - the most important entity definitions (database models) are shown in Figure 2.5. These definitions make heavy use of foreign key relations to simplify movement around the data.

The `Object` class represents the metadata for a single filesystem object, and if there is binary data associated with the object (in the case of regular files), the path of the canonical version (which can be shared between multiple objects) is stored with its SHA-1 hash as a `Prime`. These objects are created from the `DirObjects` built by the `HashTreeMaker`. CoStor generates a unique identifier for each `Object` from the SHA-1 digest of the object path concatenated with the object hash.

The hashes for `DirObjects` are generated by the `gethash` function within the `DirObject` class. This allows the hash to be returned without modifying the object instance, and inadvertently changing its hash. Object hashes are calculated from the SHA-1 digest of the binary data in the case of files, and from the digest of the concatenation of the name, path, stat metadata and the sorted string representation of its children's hashes in the case of a directory. The function is shown in Figure 2.4.

[2]PonyORM: an advanced object-relational mapper `https://ponyorm.readthedocs.io`

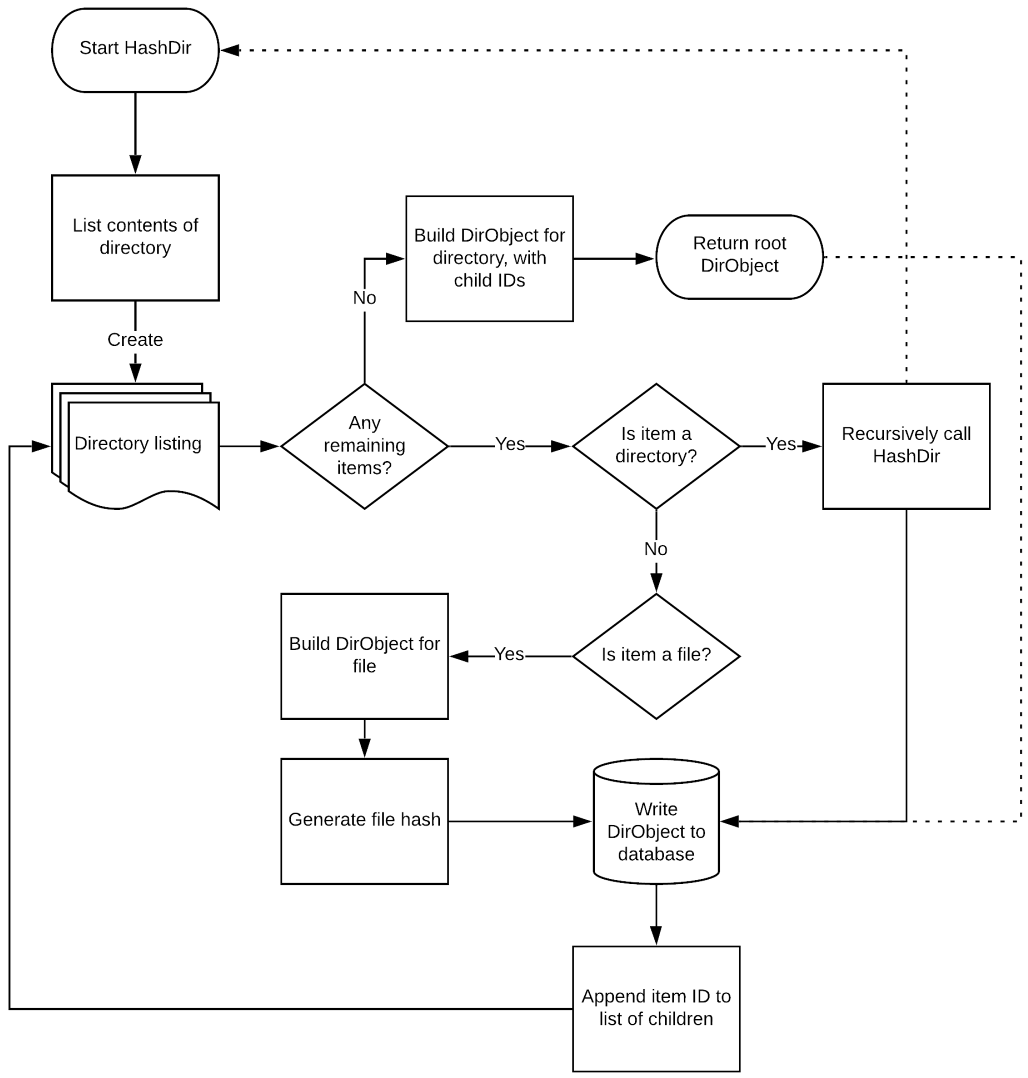


Figure 2.3: Flowchart of process for creating DB representation of directory tree

```
def gethash(self: DirObject):
if self.type is "file":
    return self.hash
elif self.type is "dir":
    shastring = self.name + self.path + str(self.stat)
    childhashes = str(sorted(self.children))
    return sha1str(shastring + childhashes)
```

Figure 2.4: Method to generate file hash from `DirObject` (costor_client/hasher.py)

```
# a specific run of this application
class Snapshot(db.Entity):
    id = PrimaryKey(int, auto=True)
    timestamp = Required(datetime)
    complete = Required(bool)
    synced = Required(bool)
    objects = Set('Object', reverse='snapshots')
    root = Required('BackupRoot', reverse='snapshots')
    topobject = Optional('Object', reverse='topobjectfor')
    parent = Optional('Snapshot', reverse='child')
    child = Optional('Snapshot', reverse='parent')
    primes = Set('Prime', reverse='snapshots')

# anything included in the backup tree
# type can be "file" or "dir" (or "sym")
class Object(db.Entity):
    id = PrimaryKey(str)  # sha(path+hash)
    hash = Required(str)
    name = Required(str)
    prime = Optional('Prime', reverse='objects')
    path = Required(str)
    type = Required(str)
    stat = Required(str)
    children = Set('Object', reverse='parent')
    parent = Optional('Object', reverse='children')
    snapshots = Set('Snapshot', reverse='objects')
    topobjectfor = Optional('Snapshot')
    depth = Required(int)

# file path to object hash mappings
class Prime(db.Entity):
    filehash = PrimaryKey(str)
    paths = Set('Path', reverse='target')
    firstseen = Required(datetime, sql_default='CURRENT_TIMESTAMP')
    lastseen = Required(datetime, sql_default='CURRENT_TIMESTAMP')
    objects = Set('Object', reverse='prime')
    snapshots = Set('Snapshot', reverse=None)
```

Figure 2.5: PonyORM database entity definitions for client (costor_client/db.py)

Chapter 3

Server and datastore implementation

3.1 Overview

CoStor server is a Django based web application, that can be deployed within Docker for easy installation and upgrades. It not only provides the backup API endpoints used by the Client software to actually push snapshot data to the server's storage, but also a web UI for management and monitoring, as well as frameworks to allow replication of backup data between instances of the server over a standard HTTPS connection.

To allow larger tasks (such as backup archive generation) to run asynchronously of the HTTP requests to the web server, it was planned to make use of a task broker such as Celery[1] , however this is not implemented within this version of CoStor.

3.2 High-level data architecture

To allow all machinery to be as generalised as possible, all possible file system objects are stored as generic `Object` objects. These can represent files, directories or symlinks. Each object holds a globally unique ID, path information, the hash of the object's contents, child and parent relations, the path and link to a `DbFile` object or ”prime” in the case that the object is a file, with the prime representing the binary data of that file on the CoStor server's filesystem.

These objects are then related to their associated ”snapshot” or backup session, backup root directory and the agent (or CoStor client identity) that the backup originated from.

3.3 Server metadata database and filestore

All data on the CoStor server is stored within the context of the Django ORM, to allow easy access to all objects using the powerful Object APIs provided by the ORM. The backup data is stored in the following database models:

[1] `http://www.celeryproject.org`

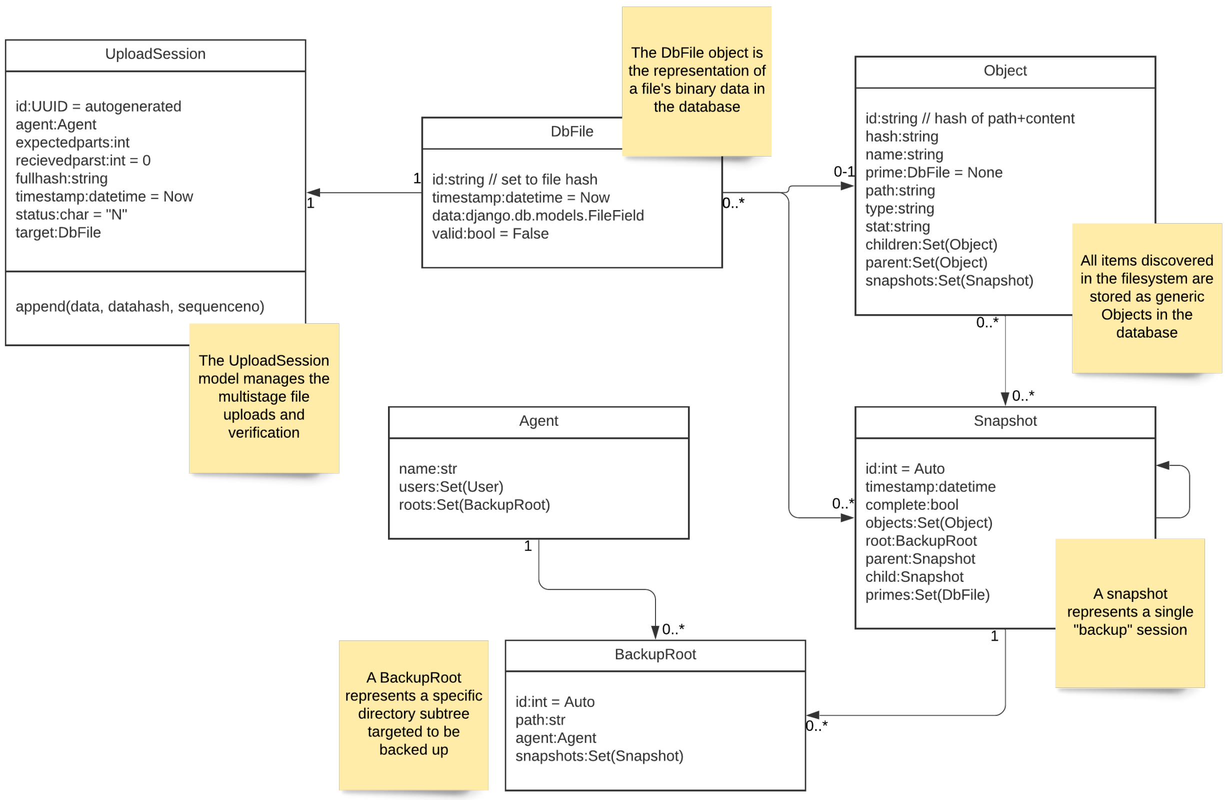


Figure 3.1: UML of Django ORM models used to manage backup data

File data is stored wrapped within a `django.orm.models.FileField` attached to the DbFile object and stored on the filesystem of the CoStor server at a location defined by the config file, with the filename being the object's ID.

3.4 Management UI

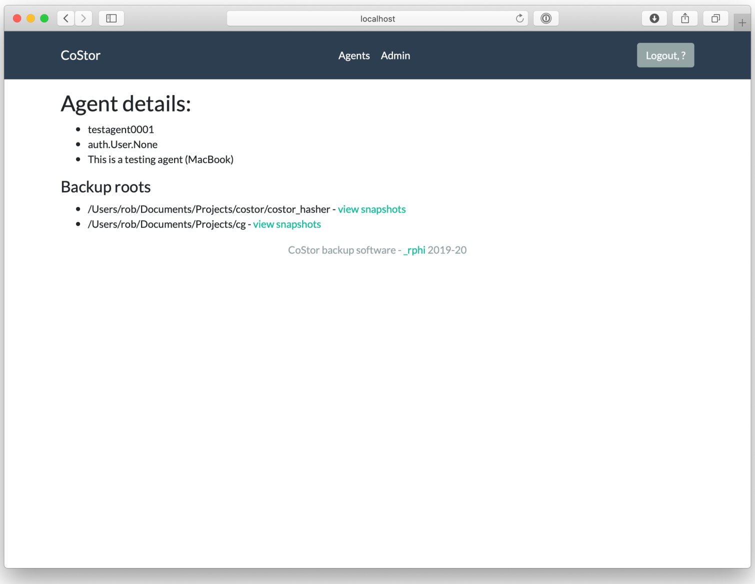
CoStor server makes use of the built in Django Admin package for management of database objects and user permissions. There is additionally a custom frontend used to browsing registered agents, their backup tasks (backup roots), snapshots and the file trees of each snapshot.

Once a snapshot has been located by the user, there is the option to download a copy of that snapshot's contents as a `tar.bz2` archive.

3.5 Restoring backups

3.5.1 Direct (local) restore

These are the most straightforward of restores, the administrator simply has to find a directory or object within the data browser interface for an agent and snapshot within the



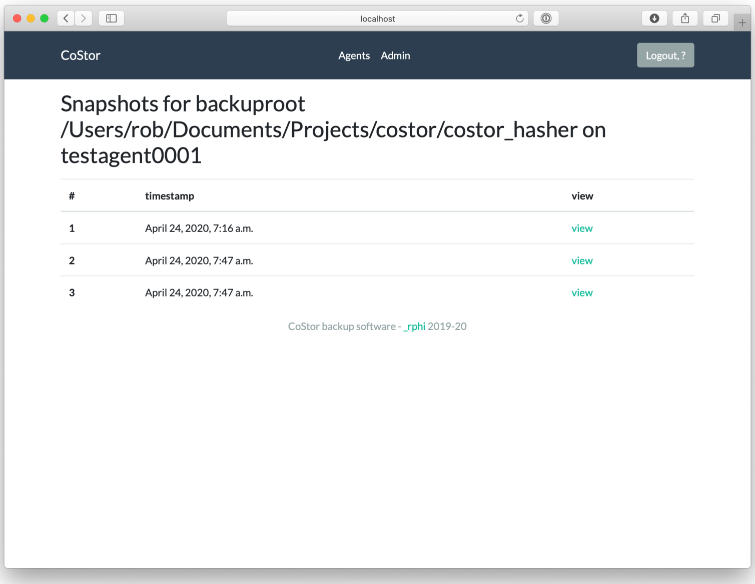
Agent details:

- testagent0001
- auth.User.None
- This is a testing agent (MacBook)

Backup roots

- /Users/rob/Documents/Projects/costor/costor_hasher - view snapshots
- /Users/rob/Documents/Projects/cg - view snapshots

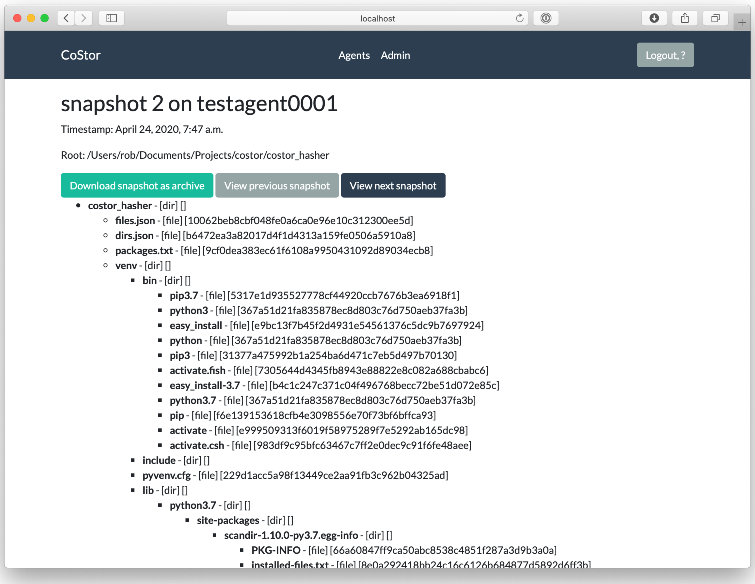
CoStor backup software - _rphi 2019-20



Snapshots for backuproot /Users/rob/Documents/Projects/costor/costor_hasher on testagent0001

#	timestamp	view
1	April 24, 2020, 7:16 a.m.	view
2	April 24, 2020, 7:47 a.m.	view
3	April 24, 2020, 7:47 a.m.	view

CoStor backup software - _rphi 2019-20



snapshot 2 on testagent0001

Timestamp: April 24, 2020, 7:47 a.m.

Root: /Users/rob/Documents/Projects/costor/costor_hasher

- **costor_hasher** - [dir] []
  - **files.json** - [file] [10062beb8cbf048fe0a6ca0e96e10c312300ee5d]
  - **dirs.json** - [file] [b6472ea3a82017d4f1d4313a159fe0506a5910a8]
  - **packages.txt** - [file] [9cf0dea383ec61f6108a9950431092d89034ecb8]
  - **venv** - [dir] []
    - **bin** - [dir] []
      - **pip3.7** - [file] [5317e1d935527778cf44920ccb7676b3ea6918f1]
      - **python3** - [file] [367a51d21fa835878ec8d803c76d750aeb37fa3b]
      - **easy_install** - [file] [e9bc13f7b45f2d4931e54561376c5dc9b7697924]
      - **python** - [file] [367a51d21fa835878ec8d803c76d750aeb37fa3b]
      - **pip3** - [file] [31377a475992b1a254ba6d471c7eb5d497b70130]
      - **activate.fish** - [file] [7305644d4345fb8943e88822e8c082a688cbabc6]
      - **easy_install-3.7** - [file] [b4c1c247c371c04f496768becc72be51d072e85c]
      - **python3.7** - [file] [367a51d21fa835878ec8d803c76d750aeb37fa3b]
      - **pip** - [file] [f6e139153618cfb4e3098556e70f73bf6bffca93]
      - **activate** - [file] [e999509313f6019f58975289f7e5292ab165dc98]
      - **activate.csh** - [file] [983df9c95bfc63467c7ff2e0dec9c91f6fe48aee]
    - **include** - [dir] []
    - **pyvenv.cfg** - [file] [229d1acc5a98f13449ce2aa91fb3c962b04325ad]
    - **lib** - [dir] []
      - **python3.7** - [dir] []
        - **site-packages** - [dir] []
          - **scandir-1.10.0-py3.7.egg-info** - [dir] []
            - **PKG-INFO** - [file] [66a60847ff9ca50abc8538c4851f287a3d9b3a0a]
            - **installed-files.txt** - [file] [8e0a292418bb24c16c6126b684877d5892d6ff3b]

Figure 3.2: Screenshots of the CoStor browser

web UI (Figure 3.2) for their site's local instance of CoStor server, before requesting a restore package be generated. CoStor then re-builds the original directory tree selected from database objects and creates an archive of the backup which can be downloaded and restored manually.

This is built by recursively traversing through the directory tree, starting from the object defined in the snapshot as the `topobject`. First the directory structure is rebuilt in a temporary directory (`/tmp/costordownload/<snapshotid>`), with file permissions, UIDs and GIDs restored from the object's `stat` attribute which was generated from the `os.stat` package when the client initially catalogued the file.

Once the tree is complete, this is archived into a `tar.bz2` package, the temporary uncompressed structure is removed and the archive is delivered to the user's browser.

This method does have some downsides:

- An in place restore (eg from a context menu) is obviously impossible, although it would require considerable research to be able to reliably hook into the supported OS' file explorer context menu APIs.
- The user cannot currently restore their own data, the archive would need to be created by the administrator.
- This archive is generated synchronously on the web server thread. Obviously with very very large backups, it would be far more sensible to move this logic into an asynchronous task runner.
- It is not possible to request a single file from a snapshot, nor can you view version history for a single file

Ultimately, it would make sense to implement a self-service restoration interface within the CoStor client, which could vastly improve the end-user experience.

3.5.2 Remote (disaster recovery) restore

NOTE: *the functionality detailed below has not been implemented in this version of CoStor, as there is no completed support for multiple-instance replication. It is however left as context to other architectural decisions made throughout CoStor*

3.5.2.1 Using offsite instance to restore individual snapshots

In the event that the instance of CoStor has failed for the site that is being backed up, it should be possible to restore a backup from another site's instance of CoStor.

1. As all metadata for replicated sites is encrypted, a valid master decryption key must be provided to the server being used to restore a backup. This is checked against the master federation database, which is replicated to all servers within the group.
2. The server then identifies which CoStor servers contain copies of the metadata databases for the requested site (using the global master federation database),

and requests that metadata be replicated to itself if the data is not already available locally.

3. Finally, the server decrypts the replicated metadata for the requested agent and snapshot, allowing the administrator to generate a restoration archive using the same method for a local restore: browsing the directory tree and selecting required sub-trees to be restored. File ”primes” or binary data blocks are decrypted on the fly before being written to the restoration archive.

3.5.2.2 Restoring entire datastore from offsite to replacement CoStor server

In the event that the hardware running CoStor has to be replaced, it should be possible to replicate all data for that site back to the new server from the replicated copies across other sites.

1. The new instance should be configured with the same ID, encryption key and access token as the original server, to re-enroll it into the network. The master federation database will then be replicated to this new server.
2. The new (replacement) server can then request that all replicated data from its site is returned to its local storage, and will decrypt each package as it arrives to rebuild the server’s original state.
3. Once all data has been restored to the server, the restoration continue as outlined in the direct (local) restore procedure.

3.6 Automated maintenance

Due to the use of the Django ORM for management of all data storage, including file storage of object primes, CoStor is very resilient to datastore inconsistencies. All database models are defined with `on_delete` actions to allow the database to automatically prune itself once a snapshot, backup root or agent is removed.

All data entry is additionally validated before being written, to ensure the database cannot contain invalid data.

Chapter 4

Multi-site replication

NOTE: *the functionality detailed in this chapter has not been implemented in this version of CoStor. It is however left as context to other architectural decisions made throughout the application*

The data to manage multi-site restores should be stored in a globally replicated database, which contains information such as the replication status of all snapshot directory trees and file ”prime” objects, anonymised agent ID to site mappings, and synchronisation job objects to track when and to where objects are replicated.

Each site would manage its own replications, and respond to replication requests from other servers within the network. All objects are identified globally by the hash of their contents, and database objects would be replicated as encrypted JSON documents. File ”primes” would transferred between sites using the large file upload protocol used by the clients (outlined in Figure 2.1)

The overall goal of the replication system would be to ensure there are at least two redundant, offsite copies of all data. This would be performed by selecting replication targets randomly, weighted by free disk space, and by incrementally scheduling the replication tasks during a configured maintenance window (normally overnight when the network is quiet).

4.1 Data distribution and sharding

Each server would choose two sites to distribute its data across to. This means that each instance of CoStor would be holding the backup data of its own site, plus compressed copies of two other sites' backups.

4.2 Replication management database

To allow all federated instances of CoStor to collaborate on managing replication tasks, and to enable restoration from federated data even if the master database for that site is unavailable (in the event of a server failure), the replication data would be synchronised across all servers.

All operations on these database tables are wrapped such that an API request is made to all other servers to update them immediately. Additionally, before a replication task commences, a full sync is requested to ensure that all servers contain all objects. To allow the servers to easily verify data integrity across nodes, all entries in this database have a UUID and are write only. When a sync is triggered, the server sends a request to all other CoStor servers with the full list of object IDs. If any IDs are missing, each server requests copies of these. If any server finds that it has an object with an ID that isn't in the list, it can assume that object has been deleted and will purge it from its own copy of the database.

This database contains information such as which backup Objects and file ”primes” (Figure 3.1) have been replicated to which locations, current status of all nodes, and basic information on which sites own which agents and snapshots. This database **must** contain all the information needed to initiate a restore in the event of a remote disaster recovery restoration (Section 3.5). As only the server to be modifying data relating to a site is the server located in that site, there is no risk of requests interfering with each other.

As almost every object in CoStor is referenced by a unique and anonymous ID, there is minimal risk in sharing this data with other servers.

4.3 Securely synchronising data

”How do we share data across sites without risking breach of confidentiality?”

To enable the backup databases to be replicated securely, replicated data is stored using a modified subset of the standard database objects used by the local backups (Figure 3.1).

The actual DbFile objects do not require changing, as the data is already stored encrypted, and thanks to the use of convergent encryption, the ID (or hash) of all of these objects is already the same as those stored from the target server's local backups. The Object class is modified such that the name, path and stat fields are stored as encrypted strings, but as the IDs, agent and snapshot IDs are unique across sites, as well as being anonymous, these foreign keys can continue to be used.

4.4 Efficiently transferring data

We also need a way to transfer that data between hosts. As we already have serialisers for the database models thanks to the Django Rest Framework, it makes sense to reuse these to transfer database objects as JSON data. This also allows us to combine multiple entries into one large request, as an entire snapshot's worth of data is still fairly small.

For file primes, if they are not already present on the target server, we can give these a quick bit of compression using something fast like LZ4[1] to compress each object before transmitting it using a modified version of the large file protocol utilised by the client, described in Figure 2.1. These can then be decompressed and added to that site's local DbFile store, where they may come in useful for local backups as well.

[1]`https://github.com/lz4/lz4`

Chapter 5

Testing and evaluation

The most important goal of a backup system is the ability to accurately first store, and then reproduce a copy of the backup target from its datastore.

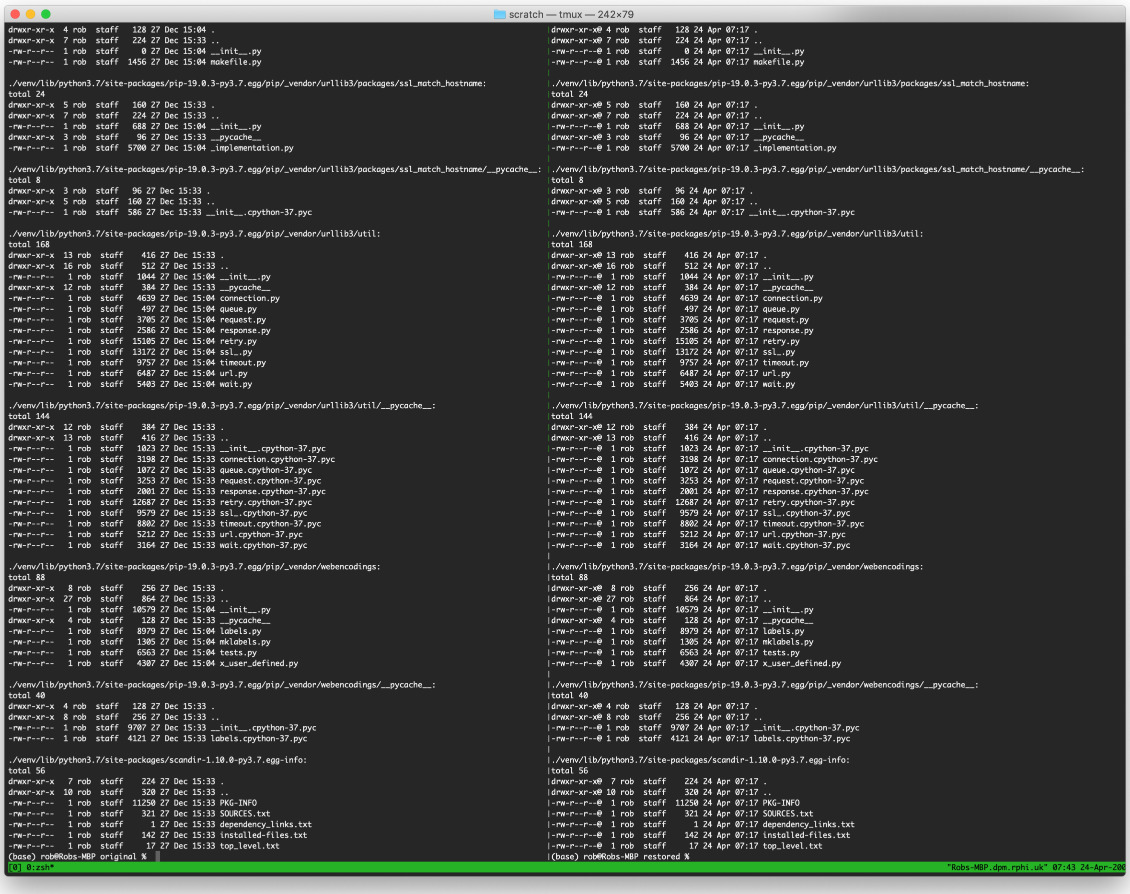
For the purpose of testing, the `cores_hasher` directory within the project repository was set as the backup root, and CoStor client was run to generate a backup to the server. This backup was downloaded as a `tar.bz2` from the CoStor management UI, and extracted alongside a copy of the `cores_hasher` test directory.

At first glance, it would be sensible to create a matching `tar.bz2` archive of the test files locally, and binary diff that against the archive given to us by CoStor. Unfortunately, this is not likely to be successful, as modification and creation timestamps are not preserved by the backup process.

The two copies of the test directory were compared using the GNU diff tool, using the `--recursive` flag to allow it to traverse all the way through the tree. Running this tool confirmed both directories were identical at a binary level, which concludes that the data must be the same.

As this is a fairly small dataset, file permissions and groups were also checked by hand, and the restored copy of the directory contained all of the correct metadata.

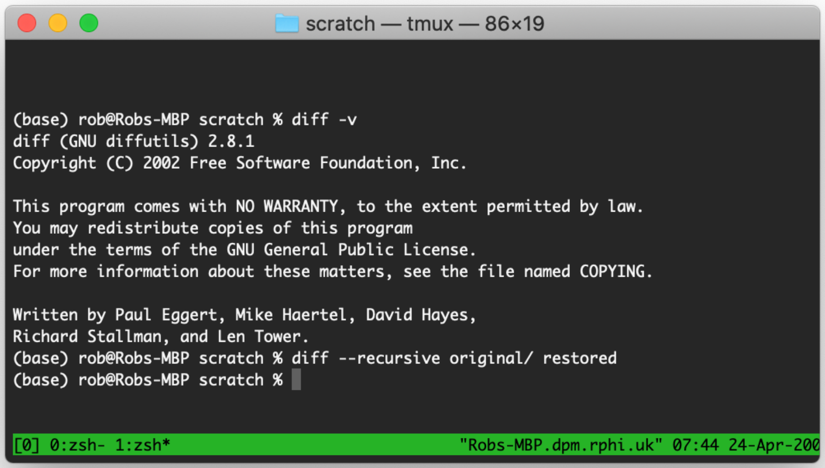
Although it would be worthwhile to perform more comprehensive testing on a larger directory tree, performing all testing had to be completed locally on the development MacBook, as any work on remote servers would be rendered fairly difficult thanks to my uplink being provided by a fairly flaky cellular data dongle. Additionally, the current synchronous archive generation method used by the web UI would unlikely be very performant with larger backups.



```
drwxr-xr-x  4 rob  staff   128 27 Dec 15:04 .
drwxr-xr-x  7 rob  staff   224 27 Dec 15:33 ..
-rw-r--r--  1 rob  staff     0 27 Dec 15:04 __init__.py
-rw-r--r--  1 rob  staff  1456 27 Dec 15:04 makefile.py

./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/packages/ssl_match_hostname:
total 24
drwxr-xr-x  5 rob  staff   160 27 Dec 15:33 .
drwxr-xr-x  7 rob  staff   224 27 Dec 15:33 ..
-rw-r--r--  1 rob  staff   688 27 Dec 15:04 __init__.py
drwxr-xr-x  3 rob  staff    96 27 Dec 15:33 __pycache__
-rw-r--r--  1 rob  staff  5700 27 Dec 15:04 _implementation.py

./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/packages/ssl_match_hostname/__pycache__:
total 8
drwxr-xr-x  3 rob  staff   96 27 Dec 15:33 .
drwxr-xr-x  5 rob  staff  160 27 Dec 15:33 ..
-rw-r--r--  1 rob  staff  586 27 Dec 15:33 __init__.cpython-37.pyc

./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/util:
total 168
drwxr-xr-x  13 rob  staff    416 27 Dec 15:33 .
drwxr-xr-x  16 rob  staff    512 27 Dec 15:33 ..
-rw-r--r--   1 rob  staff   1044 27 Dec 15:04 __init__.py
drwxr-xr-x  12 rob  staff    384 27 Dec 15:33 __pycache__
-rw-r--r--   1 rob  staff   4639 27 Dec 15:04 connection.py
-rw-r--r--   1 rob  staff    497 27 Dec 15:04 queue.py
-rw-r--r--   1 rob  staff   3705 27 Dec 15:04 request.py
-rw-r--r--   1 rob  staff   2586 27 Dec 15:04 response.py
-rw-r--r--   1 rob  staff  15105 27 Dec 15:04 retry.py
-rw-r--r--   1 rob  staff  13172 27 Dec 15:04 ssl_.py
-rw-r--r--   1 rob  staff   9757 27 Dec 15:04 timeout.py
-rw-r--r--   1 rob  staff   6487 27 Dec 15:04 url.py
-rw-r--r--   1 rob  staff   5403 27 Dec 15:04 wait.py

./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/util/__pycache__:
total 144
drwxr-xr-x  12 rob  staff    384 27 Dec 15:33 .
drwxr-xr-x  13 rob  staff    416 27 Dec 15:33 ..
-rw-r--r--   1 rob  staff   1023 27 Dec 15:33 __init__.cpython-37.pyc
-rw-r--r--   1 rob  staff   3198 27 Dec 15:33 connection.cpython-37.pyc
-rw-r--r--   1 rob  staff   1072 27 Dec 15:33 queue.cpython-37.pyc
-rw-r--r--   1 rob  staff   3253 27 Dec 15:33 request.cpython-37.pyc
-rw-r--r--   1 rob  staff   2001 27 Dec 15:33 response.cpython-37.pyc
-rw-r--r--   1 rob  staff  12687 27 Dec 15:33 retry.cpython-37.pyc
-rw-r--r--   1 rob  staff   9579 27 Dec 15:33 ssl_.cpython-37.pyc
-rw-r--r--   1 rob  staff   8802 27 Dec 15:33 timeout.cpython-37.pyc
-rw-r--r--   1 rob  staff   5212 27 Dec 15:33 url.cpython-37.pyc
-rw-r--r--   1 rob  staff   3164 27 Dec 15:33 wait.cpython-37.pyc

./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/webencodings:
total 88
drwxr-xr-x   8 rob  staff    256 27 Dec 15:33 .
drwxr-xr-x  27 rob  staff    864 27 Dec 15:33 ..
-rw-r--r--   1 rob  staff  10579 27 Dec 15:04 __init__.py
drwxr-xr-x   4 rob  staff    128 27 Dec 15:33 __pycache__
-rw-r--r--   1 rob  staff   8979 27 Dec 15:04 labels.py
-rw-r--r--   1 rob  staff   1305 27 Dec 15:04 mklabels.py
-rw-r--r--   1 rob  staff   6563 27 Dec 15:04 tests.py
-rw-r--r--   1 rob  staff   4307 27 Dec 15:04 x_user_defined.py

./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/webencodings/__pycache__:
total 40
drwxr-xr-x  4 rob  staff   128 27 Dec 15:33 .
drwxr-xr-x  8 rob  staff   256 27 Dec 15:33 ..
-rw-r--r--  1 rob  staff  9707 27 Dec 15:33 __init__.cpython-37.pyc
-rw-r--r--  1 rob  staff  4121 27 Dec 15:33 labels.cpython-37.pyc

./venv/lib/python3.7/site-packages/scandir-1.10.0-py3.7.egg-info:
total 56
drwxr-xr-x   7 rob  staff    224 27 Dec 15:33 .
drwxr-xr-x  10 rob  staff    320 27 Dec 15:33 ..
-rw-r--r--   1 rob  staff  11250 27 Dec 15:33 PKG-INFO
-rw-r--r--   1 rob  staff    321 27 Dec 15:33 SOURCES.txt
-rw-r--r--   1 rob  staff      1 27 Dec 15:33 dependency_links.txt
-rw-r--r--   1 rob  staff    142 27 Dec 15:33 installed-files.txt
-rw-r--r--   1 rob  staff     17 27 Dec 15:33 top_level.txt
(base) rob@Robs-MBP original %
[0] 0:zsh*
```

```
|drwxr-xr-x@ 4 rob  staff   128 24 Apr 07:17 .
|drwxr-xr-x@ 7 rob  staff   224 24 Apr 07:17 ..
|-rw-r--r--@ 1 rob  staff     0 24 Apr 07:17 __init__.py
|-rw-r--r--@ 1 rob  staff  1456 24 Apr 07:17 makefile.py
|
|./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/packages/ssl_match_hostname:
|total 24
|drwxr-xr-x@ 5 rob  staff   160 24 Apr 07:17 .
|drwxr-xr-x@ 7 rob  staff   224 24 Apr 07:17 ..
|-rw-r--r--@ 1 rob  staff   688 24 Apr 07:17 __init__.py
|drwxr-xr-x@ 3 rob  staff    96 24 Apr 07:17 __pycache__
|-rw-r--r--@ 1 rob  staff  5700 24 Apr 07:17 _implementation.py
|
|./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/packages/ssl_match_hostname/__pycache__:
|total 8
|drwxr-xr-x@ 3 rob  staff   96 24 Apr 07:17 .
|drwxr-xr-x@ 5 rob  staff  160 24 Apr 07:17 ..
|-rw-r--r--@ 1 rob  staff  586 24 Apr 07:17 __init__.cpython-37.pyc
|
|./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/util:
|total 168
|drwxr-xr-x@ 13 rob  staff    416 24 Apr 07:17 .
|drwxr-xr-x@ 16 rob  staff    512 24 Apr 07:17 ..
|-rw-r--r--@  1 rob  staff   1044 24 Apr 07:17 __init__.py
|drwxr-xr-x@ 12 rob  staff    384 24 Apr 07:17 __pycache__
|-rw-r--r--@  1 rob  staff   4639 24 Apr 07:17 connection.py
|-rw-r--r--@  1 rob  staff    497 24 Apr 07:17 queue.py
|-rw-r--r--@  1 rob  staff   3705 24 Apr 07:17 request.py
|-rw-r--r--@  1 rob  staff   2586 24 Apr 07:17 response.py
|-rw-r--r--@  1 rob  staff  15105 24 Apr 07:17 retry.py
|-rw-r--r--@  1 rob  staff  13172 24 Apr 07:17 ssl_.py
|-rw-r--r--@  1 rob  staff   9757 24 Apr 07:17 timeout.py
|-rw-r--r--@  1 rob  staff   6487 24 Apr 07:17 url.py
|-rw-r--r--@  1 rob  staff   5403 24 Apr 07:17 wait.py
|
|./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/urllib3/util/__pycache__:
|total 144
|drwxr-xr-x@ 12 rob  staff    384 24 Apr 07:17 .
|drwxr-xr-x@ 13 rob  staff    416 24 Apr 07:17 ..
|-rw-r--r--@  1 rob  staff   1023 24 Apr 07:17 __init__.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   3198 24 Apr 07:17 connection.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   1072 24 Apr 07:17 queue.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   3253 24 Apr 07:17 request.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   2001 24 Apr 07:17 response.cpython-37.pyc
|-rw-r--r--@  1 rob  staff  12687 24 Apr 07:17 retry.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   9579 24 Apr 07:17 ssl_.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   8802 24 Apr 07:17 timeout.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   5212 24 Apr 07:17 url.cpython-37.pyc
|-rw-r--r--@  1 rob  staff   3164 24 Apr 07:17 wait.cpython-37.pyc
|
|./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/webencodings:
|total 88
|drwxr-xr-x@  8 rob  staff    256 24 Apr 07:17 .
|drwxr-xr-x@ 27 rob  staff    864 24 Apr 07:17 ..
|-rw-r--r--@  1 rob  staff  10579 24 Apr 07:17 __init__.py
|drwxr-xr-x@  4 rob  staff    128 24 Apr 07:17 __pycache__
|-rw-r--r--@  1 rob  staff   8979 24 Apr 07:17 labels.py
|-rw-r--r--@  1 rob  staff   1305 24 Apr 07:17 mklabels.py
|-rw-r--r--@  1 rob  staff   6563 24 Apr 07:17 tests.py
|-rw-r--r--@  1 rob  staff   4307 24 Apr 07:17 x_user_defined.py
|
|./venv/lib/python3.7/site-packages/pip-19.0.3-py3.7.egg/pip/_vendor/webencodings/__pycache__:
|total 40
|drwxr-xr-x@ 4 rob  staff   128 24 Apr 07:17 .
|drwxr-xr-x@ 8 rob  staff   256 24 Apr 07:17 ..
|-rw-r--r--@ 1 rob  staff  9707 24 Apr 07:17 __init__.cpython-37.pyc
|-rw-r--r--@ 1 rob  staff  4121 24 Apr 07:17 labels.cpython-37.pyc
|
|./venv/lib/python3.7/site-packages/scandir-1.10.0-py3.7.egg-info:
|total 56
|drwxr-xr-x@  7 rob  staff    224 24 Apr 07:17 .
|drwxr-xr-x@ 10 rob  staff    320 24 Apr 07:17 ..
|-rw-r--r--@  1 rob  staff  11250 24 Apr 07:17 PKG-INFO
|-rw-r--r--@  1 rob  staff    321 24 Apr 07:17 SOURCES.txt
|-rw-r--r--@  1 rob  staff      1 24 Apr 07:17 dependency_links.txt
|-rw-r--r--@  1 rob  staff    142 24 Apr 07:17 installed-files.txt
|-rw-r--r--@  1 rob  staff     17 24 Apr 07:17 top_level.txt
|(base) rob@Robs-MBP restored %
                                        "Robs-MBP.dpm.rphi.uk" 07:43 24-Apr-200
```

Figure 5.1: Output of `ls -alR` for the original and restored versions of the test directory



```
(base) rob@Robs-MBP scratch % diff -v
diff (GNU diffutils) 2.8.1
Copyright (C) 2002 Free Software Foundation, Inc.

This program comes with NO WARRANTY, to the extent permitted by law.
You may redistribute copies of this program
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.

Written by Paul Eggert, Mike Haertel, David Hayes,
Richard Stallman, and Len Tower.
(base) rob@Robs-MBP scratch % diff --recursive original/ restored
(base) rob@Robs-MBP scratch %

[0] 0:zsh- 1:zsh*                    "Robs-MBP.dpm.rphi.uk" 07:44 24-Apr-200
```

Figure 5.2: Output of the `diff --recursive` command on the original and restored versions of the test directory

Bibliography

[1] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in *Proceedings of the Large Installations Systems Administration (LISA) Conference*. Berkeley, CA: Usenix Association, November 2010. [Online]. Available: http://homepages.inf.ed.ac.uk/dcspaul/publications/lisa2010.pdf

[2] Syncthing website. [Online]. Available: https://syncthing.net

[3] Urbackup website. [Online]. Available: https://www.urbackup.org/index.html

[4] (2020, January) Hermes project on github. [Online]. Available: https://github.com/Hermes/hermes

[5] Bacula website. [Online]. Available: https://www.bacula.org

[6] Gluster project homepage. [Online]. Available: https://www.gluster.org

[7] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Apress, 2014. [Online]. Available: https://git-scm.com/book/en/v2

[8] MikeRayMSFT, "Create a full database backup - sql server," Dec 2019. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/create-a-full-database-backup-sql-server?view=sql-server-ver15