File versioning systems

It can be very useful to keep earlier versions of files.

- We can recover from mistakes.
- We can restore damaged files.
- We can compare versions to see the changes made.
  - Useful for security purposes (self-securing storage)
  - Also useful for other purposes – e.g. working on a project with others and you want to see the changes they made.

Similar to code management services like CVS.

- Sometimes we want to use earlier versions and still hold on to the recent versions.

Can be done in a variety of ways but all require extra disk space.

Methods of versioning

A new version could be created when the file is closed or saved.

A new version could be created after every modification – known as comprehensive versioning.

Obviously a lot more versions.

Either way we can -

- keep complete copies of all previous versions

very space intensive

but fast to retrieve/recover

- keep a journal (or log) of changes

the journal keeps a record of the changes between two versions

retrieving requires work to reproduce earlier versions

- keep a tree with all data

finding any version takes the same amount of time

can be slow for current version if the tree is big

Example

e.g.

a file with the contents:

"Dear Mum, I hope you are well."

gets modified and saved as:

"Dear Mum, I am doing really well in my Operating System course. I hope you are well."

then as:

"Dear Mum, I thought I was doing really well in my Operating System course until I sat the test. I hope you are well."

Log version

Original version was

```
Dear Mum, I hope you are well.
```

changes to get to version 2

`11i54` (54 chars inserted at position 11)

changes to get to version 3

`13d` (a deletion at position 13)

`am` (the deleted data, this must be kept)

```
13i13
74i21
```

The current version is always stored.

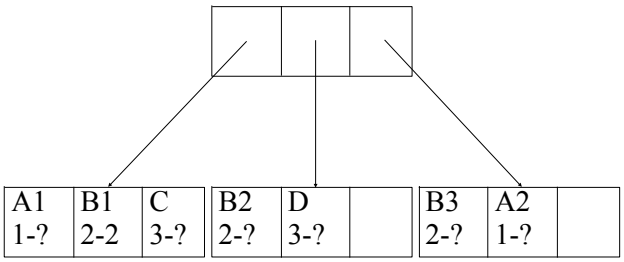
```
Dear Mum, I thought I was doing really
well in my Operating System course
until I sat the test. I hope you are
well.
```

To get previous versions have to go backwards through the log.

If we want to be able to roll forward from a checkpoint we need the new data in lines like `13i13`.

Multiversion B-tree

A1: Dear Mum, I_
A2: hope you are well.
B1: am
B2: _doing really well in my Operating System course
B3: ._I_
C: thought I was
D: _until I sat the test



The version ranges (2-?) show which version the leaf is valid for.

? means up to the present.

Advantages and disadvantages

Log system

- very compact
- access to the current version is the same as without versioning
- slow to revert to previous versions especially if there are many versions
- can use checkpoints to improve this, but this adds considerably to the space requirement

Tree system

- very compact
- quick to revert to any previous version
- if there are lots of versions the tree can be big and then access to the current version will be a little slow
- can keep a separate copy of the current version, this also adds to the space requirement

No method works well if the data between versions is completely different. We are stuck with having to keep complete versions.

VMS versions

When a file is closed VMS checks the number of versions. If the number is greater than the maximum number then the oldest version is discarded.

Windows XP onwards

It takes a checkpoint (restore points) of important system files on a regular basis.
daily
on installation of new drivers and applications

NTFS maintains a log of all changes to metadata, along with redo,undo information and whether the change was committed.

So it can recover all metadata to a consistent state after a crash (but not all data).

Windows and Mac

Windows - Volume Shadow Copy

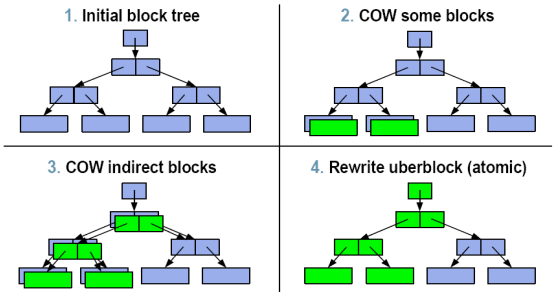
- Keeps copies of files on volumes which have the service turned on
- Also used to create restore points
- Works at the block level - only modified blocks are copied
- Typically made once a day
- Users cannot trigger new versions of individual files
- Users can access versions from Previous Versions tab of file properties
- Not available at GUI level in Windows 8 but still in Windows Server

OS X Lion onwards - Versions

- Auto saves individual files (if enabled by the application)
- Works on chunks (intelligently determined by content) - only modified chunks are copied
- Typically made every hour (autosave works every 5 minutes or during pauses)
- Users can trigger new versions for individual files
- Accessed differently after Mavericks

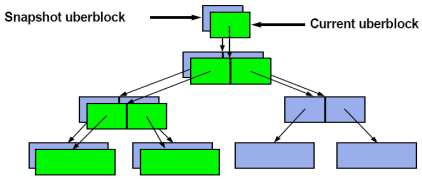
ZFS snapshots

Copy-On-Write Transactions



Bonus: Constant-Time Snapshots

- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!



Pruning

All conventional versioning systems use pruning to keep the amount of data stored under control.

Different heuristics

- a fixed number of versions
- treat some changes as more important than others
- "observe" user behaviour, e.g., most often accessed
- the user has to explicitly request a version be held

snapshot systems – keep versions of files at particular times

- only keep versions for a small number of files

Self-securing storage

All metadata, directories and critical files (OS files) are kept on a versioning system.

Any intrusion (that uses files) can be tracked because the intruder cannot erase changes they have made to the system.

We need to maintain all versions between checks for intrusion.

This is referred to as the detection window.

If the system is unable to keep enough versions we signal an alarm.

Either something has gone wrong, in the sense of not enough space allocated for a normal amount of usage.

Or someone is trying to force a pruning to hide their tracks.

Distributed File Systems

A Distributed File System (DFS) is a file system which has data stored in several different sites or hosts (computers and associated devices) on a network.

Advantages

- greater amount of storage
- greater flexibility for administration and sharing purposes
  - users can log on to any machine and have access to all of their files
  - files can be stored close to where they are normally used but are still available elsewhere
- can replicate information for greater reliability
  - if a site goes down the files may still be accessible from another location

Accessing Remote Files

Different approaches to providing access to files over networks.

Remote file transfer approach

- No requirement for the machines to even be running the same operating system.
- A user can explicitly connect to another machine on the network and download files. e.g. ftp
- Can't directly use a file on the remote site.
- We end up with multiple copies and no method of maintaining consistency.
- The user must know exactly where a file is (including the host).

Direct access using explicit site names

- Each file is prefixed with a site identifier e.g. cs26.auckland.ac.nz:/home/robert-s/310exam
- Can use a file directly from another site
  but it is usually automatically copied to and fro.
- The user still must know exactly where a file is located.
- No replication possible.

Better methods

It is better if we don't have to specify the host name when accessing remote files.

Keep information on each machine pointing to the machine where the files are actually stored.

- files can be used directly on the remote host
- the user sees no difference between accessing remote and local files
- remote files may not be visible from all machines on the network, even if they are, they may have different pathnames
- moving remote directories entails changes on all local machines

Keep a server(s) with location information and associate a standard directory base with all remote files.

- direct use and the same view of the file system regardless of where you are logged on from
- can move files without any information needing to be sent to local machines
- can replicate files

Transparency

An ideal – the distributed system should look like a single machine and associated files.

It is called transparency because the user should not be able to see the differences (and complications).

Location transparency

No (obvious) connection between the name of a resource (file) and its position on the system.

Migration transparency

Resources (files) can move around the system without programs needing to be changed (or stopped and restarted).

This is similar to what the textbook calls *Location independence* - the name doesn't have to be changed with the location changes.

It needs location transparency to implement.

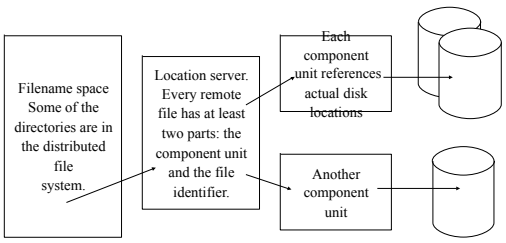
Collections of files

All common distributed file systems group files into collections for simplicity and administration purposes.

The collections (component units in the textbook) are commonly subtrees stemming from particular directories.

So the subtrees are shared and moved and replicated together.

If we are going to transparently migrate component units we need to have a structure something like this:



Using remote files

Once we have discovered where a file is we have to move all data accessed from the file over the network.

This is expensive and has problems with consistency.

We could:

- send only required blocks back and forth between the server and the client (**remote service**), every file access results in messages across the network
- keep copying large chunks when required and **cache** them at the client
- copy complete files when accessed and copy back when done (only if written to)

Caching

Blocks of files are cached locally.

All accesses come from the cache.

If a block is not in the cache it is requested from the server and then cached.

Old blocks can be replaced using Least Recently Used algorithm.

If we modify data in the cache -

- when do we send the information back to the server?
  - write-through – every write requires the block to be sent back to the server
  - delayed-write – send the block to the server at a later time (check every 30 secs, or when the file is closed or when the cached block is needed for another block)
- how do we cope with writing to the cache when other processes (on other sites) also have the file open?

Pros and Cons

caching - usually faster, more efficient, scales better than remote service

remote service - simpler to implement because of no consistency problem, uses less local memory (primary or secondary), matches local file access

Some systems use both schemes, basically providing a remote service solution but with some caching for efficiency reasons.

Consistency semantics

The way changes in data get distributed between processes accessing the same files is known as consistency semantics.

Two major types:

- UNIX semantics – any change made by any process is immediately visible to any other process.
- Session semantics – the process gets a copy of the file when it is opened and changes are not visible to other processes until a file is closed.

Both can be worked with but programmers need to know which is used on the system they are programming.

Before next time

Read from the textbook

12.8 NFS

17.9 Distributed File Systems

17.1 Advantages of Distributed Systems

17.2 Types of Network-based Operating Systems