Reconnecting Disconnected AFS

Simon Wilkinson <simon@sxw.org.uk>
School of Informatics, University of Edinburgh

The University of Edinburgh is a charitable body, registered in Scotland, with registration number SC005336.

Introduction

- History
- Overview, and introduction to the cache manager
- Code Archaelogy
- Implemenation
- Future directions

History

- Disconnected AFS was originally implemented by a group at the University of Michigan against the Transarc AFS codebase

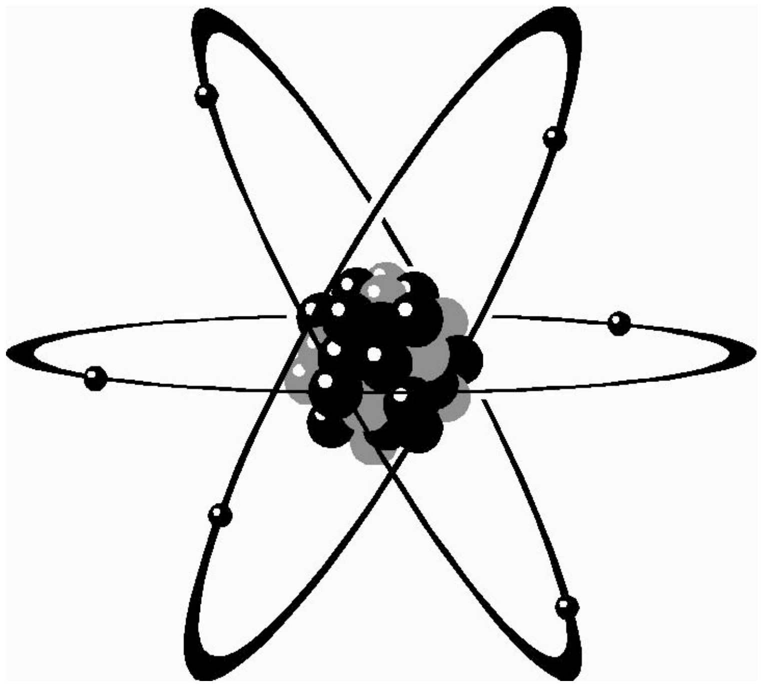
- Their implementation is documented in
  L.B. Huston, and P. Honeyman. “Disconnected Operation for AFS”, *Proceedings of the USENIX Mobile and Location- Independent Computing Symposium, August, 1993*
  `http://www.citi.umich.edu/techreports/reports/citi-tr-93-3.ps.gz`

General Principles

- Use the data in the client's cache to allow a client to continue to work when it loses access to the fileservers
- Record any changes that the client makes whilst offline
- Replay these changes to the server when the client comes back online

The cache manager from a million feet

- A very high level idea of what’s going on in the cache manager helps when explaining the issues



- This is is a very rough overview. It’s very high level, and contains a number of white lies.

Files, directories and vnodes

- A vnode is the kernel structure that represents an object (either a file, or a directory)
- Defined by the core kernel - AFS adds some additional entries
- I’ll talk mainly about files - feel free to substitute
- On Linux the vnode belongs to the kernel, not the filesystem module.

Caches I

- The client actually maintains a number of different caches

- DCache
  - The dcache contains data for files you have accessed
  - Files are split into multiple chunks
  - Cache can be either disk or memory backed. Disk backed caches persist across reboots

- VCache
  - The vcache contains file metadata
  - Held in memory
  - Entries for a file may exist in the dcache, but not in the vcache

Caches II

- volume cache
  - Maintains details of visited volumes
  - Held in memory, but backed to disk
  - Disk copy deleted on restart
- cell cache
  - Memory only
- ... and some others

Archaeology

- Derrick committed a version of the Michigan code to OpenAFS CVS in the `disconnected` branch
- Severely bit rotted
- Doesn’t build - major architectural changes have occurred since it was written
- Doesn’t implement things in the best way for the current code base

Implementation : Take 1

- **Step 1**: Take Michigan code, and port to current AFS codebase
- **Step 2**: Iterate step 1 in spare time for a number of months
- **Step 3**: Admit defeat

Implementation : Take 2

- Start from scratch, using the Michigan code as a guide
- Initially build a read-only disconnected client
  (Arla did this years ago)
- When the client's offline, give access to data in the cache
- Deny requests
  - Which require write access
  - Which can't be satisfied from the cache
- Sounds simple? Well...

Read only Implementation

- This is done, and available now
- Worth considering the issues encountered, as they'll also haunt the read-write implementation
  - Manual connection
  - Cache recency
  - Locking
  - Access Control
  - Persistence

Manual Connection

- Require human intervention to switch state
- ‘fs discon’ command, which must be executed by root
- Have to make sure file system is quiet when this occurs!
- Doing this is perhaps less usable than automatic switches, but it avoids significant UI problems

Cache recency

- Once a piece of data hits your cache, it stays there until the cache fills.
- No guarantee of recency once the callback expires
- We make all files available, regardless of whether they had a valid callback when the machine disconnected
- It’s up to user space to update all required files before it disconnects

Locking

- Read-only usage can still result in fileserver contact
- When we’re disconnected, we can only say ‘yes’
  - Can enforce locks between process on a local machine - some platforms give us this for free.
- When we reconnect, we need to ask the fileserver for any locks that we still expect to hold
- What do we do if it refuses?
  - Invalidate the current filehandle

Access control

- When we lose the network, we lose the ability to make fully informed access control decisions
- Two options:
  - It’s your disk, you can read it, do as you like
  - You can have any access that you had whilst connected
- Michigan did the first, we’re doing the second

Access control complications

- Cache manager stores previous accesses using the PAG number
- This doesn’t persist across reboots
- Nor can you explicitly request a PAG number when you change PAGs (because this would be a security hole)

Persistence

- Not all of the required data persists across reboots
- In order to survive reboots, we must be able to store this data to local disk
- Some pieces of data can’t be correctly reloaded (cached access rights, for example), and must be rewritten on reboot.

Moving on to read write

- Dragos Tatulea is working, as part of Google Summer of Code, on adding read write support
- Using a design thrashed out between Jeff Altman and myself
- Differs from the original Michigan design in a number of important ways

The original approach: journalling

- Record every change made by a client into a journal
- When we reconnect, replay that journal back to the fileserver
- Issues
  - Duplication (entries in both the AFS cache, and the journal)
  - Redundant entries (create a file, then delete, gives 2 entries)
- There was an optional journal optimiser that sought to resolve the second issue

The new way: Utilise the cache

- Local cache already has to store all of the information.
- Make *all* changes into local cache.
  (Some operations currently get the fileserver to make the change, then read it back)
- Flag cache entries as being dirty (and ensure they're flushed to disk)
- Replay all dirty entries to the fileserver when we reconnect. Do so with a lock on every entry we replay
  (We'll talk about conflict resolution shortly)

Issues

- No ordering
- No separation

Conflict resolution

- All this is fine when there’s only 1 client. The real world isn’t that simple.
- We have to be able to resolve conflicts. These occur when
  - Client A goes disconnected
  - Client A changes file Z (and the change is cached)
  - Client B changes file Z (on the fileserver)
  - Client A reconnects
- AFS data versions allow us to identify when conflicts occur

Resolution strategies

- Many simple resolution strategies
  - Server wins
  - Client wins
  - Last writer wins
  - Ask

$\left.\begin{array}{l}\text{Server wins}\\\text{Client wins}\\\text{Last writer wins}\end{array}\right\}$ *dataloss!*

- Options exist for what to do with the rejected files
  - Write them to local storage
  - Write them to an alternate name in AFS (quota permitting)

- More complex strategies may be possible, but all require file-specific knowledge

There is no perfect solution

... but there should be a choice of imperfect ones

Conflict resolution for directories

- As I mentioned, directories are a special kind of file
- ... but AFS knows the format of a directory, so we can resolve many conflicts
- ... given a common ancestor
- Disconnected client needs 2 copies of modified dirs
  - The current copy, that's in active use
  - The copy it had when it disconnected (the common ancestor)

More Replay Issues

- We’ve assumed a single set of tokens
- How do we handle machines with multiple users writing with different tokens?
- Have to use multiple tokens for replay, but also know which changes happened with which token
- It’s unlikely that there will be any movement on this, this year.

Filesets

- We want to be able to allow the user to define things for particular sets of files:
  - *Pinning* - files which are to be kept in the cache so they're available when disconnected
  - *Access* - whether a file should be available r/o or r/w when disconnected
  - *Resolution policy* - what to do should a conflict arise

Future Developments

- Automatic connection detection & per volume disconnection
  - When a volume goes away, switch to disconnected mode for its contents
  - When a volume comes back, replay any changes to it

- This raises big usability issues
  - How do you notify of replay conflicts?

More Future developments

- User interface
  - At the moment, there is no user interface for any of this, beyond the ‘fs’ command
  - GUIs would help, especially if we do non-commanded connection and disconnection
  - On multi-user machines, where to display the GUI is a complex question

Trying it out

- Patch for readonly operation available now
- Dragos will be providing regular updates of the rw code
- Try it out!
  - I promise I won’t eat your data... Much

Questions?