AFS Under the Hood

But not Inside the Carburetor

It has all your data already; you might as well know how it works.

Tim Abbott and Mitch Berger
{tabbott,mitchb}@mit.edu

SIPB Cluedump Series – September 18, 2006

Copyright © 2006 Timothy Abbott and Mitchell Berger. Some rights reserved. This work is licensed under the Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

History of AFS

- Developed at CMU as part of the Andrew Project (this is where *vice*, *virtue*, and *venus* come from) around 1985. Stood for *Andrew File System*.
- Turned into a commercial product by Transarc Corp. around 1990 (AFS no longer an acronym).
- Transarc bought by IBM. IBM became more interested in developing DFS and forked the source as OpenAFS in September of 2000.

Why not just keep using NFS?

- Network File System (Sun)
- Kerberos authentication hacked in by MIT
- Scales horribly with lots of clients
- No replication - if server goes down, you lose
- No local disk caching on client machines
- No integrated backups
- Must know name of server (Athena developed the locker model to get around this)

Features of AFS

- Location Transparency (you don’t need to know the address of the fileserver)
- Caching
- Secure Authentication
- Scalability (designed to perform well with 200:1 client:server ratio)
- Replication of important or commonly used data
- Encryption (relatively recent, and only DES)
- Uniform namespace (path to a given file is the same regardless of client machine)
- Easy to administer once you get the hang of it

Cells

- A cell refers to a site running AFS (that is, a collection of servers and clients)
- Cells are independently administered (ie: both the athena.mit.edu and sipb.mit.edu cells are at MIT, but are administered by different people)
- Cell names are formatted like, and usually related to, the DNS domain name of the site running the cell
- Users belong to a cell in that they have an account there, but they can belong to many cells
- Machines have a default cell for users on that machine

Volumes

- A volume is a container for a set of files. They can be any size, but must reside in a single partition on a fileserver
- Volumes can be stored on one server or replicated on several of them.
- Each volume appears at zero or more places in the /afs hierarchy when it is mounted there.
- Volumes are how resources are managed (each volume has a quota).
- In most installations, volumes are named with a prefix that specifies what they contain. For example, user.tabbott is Tim's home directory. contrib.consult is the consult locker. project.outland in the sipb.mit.edu cell is the outland locker.
- Volume names cannot exceed 22 characters, and when the suffixes *.readonly* or *.backup* are added to them, they have special meaning.

Types of AFS server machines

- File server – actually handles the files and volumes
- Database server – handles information about authentication, permissions, backups, and where volumes are stored
- Update server – helps to update binaries and config files on all of the cell's servers

File servers

- An AFS file server runs three important interrelated processes: *fileserver*, *volserver*, and *salvager*
- *fileserver* works much like a local filesystem (maintains a hierarchy, fetches and writes data, maintains advisory locks, deals with symlinks, checks permissions via cooperation with a protection database server)
- *volserver* handles volumes as a whole (keeps track of quota, ownership, mountpoints, usage statistics, creates, deletes, and dumps volumes, etc.)
- *salvager* repairs damage done to volumes and files due to crashes of *fileserver* or *volserver* much as fsck does for local filesystems
- Users and administrators interact with *fileserver* and *volserver* primarily with the *fs* command. Usually, *salvager*'s operation is handled automatically by the *Basic OverSeer Server*

Database servers

- An AFS database server runs four separate autonomous processes
  - Volume Location Server
  - Backup Server
  - Authentication Server
  - Protection Server
- Cells can have a single database server or multiple database servers (three is a common choice)
- These servers are critical – a cell will be unusable without them even if all file servers in the cell are working properly
- To ensure high availability, all the databases are replicated across all the database servers via a DB technology called *Ubik*
- Database servers for a cell are listed in either the *CellServDB* file or in DNS AFSDB records. These are the only servers a client needs to know about to use a cell.

Volume Location Server

- This is the *vlserver* process
- It maintains the *VLDB*
- Keeps track of which file server(s) store each volume
- Provides client cache managers with info about where to find the volumes
- Interaction with it is primarily via the *vos* command suite

Backup Server

- This is the *buserver* process
- It maintains the *BUDB*
- Keeps track of information for the AFS backup system
- Interaction with it is primarily via the *backup* command suite

Authentication Server

- This is the *kaserver* process
- It maintains the *KADB*
- Keeps track of secret keys for both the cell servers and users
- Provides clients with *tokens* to authenticate them to file servers
- Based on Kerberos v4 (the “k” is for Kerberos, not kwality)
- Not necessary if the site already runs a Kerberos installation (MIT doesn’t use it)
- Interaction with it is primarily via the *kas* command suite

Protection Server

- This is the *ptserver* process
- It maintains the *PRDB*
- Keeps track of users of the cell and group membership
- Maps usernames (found in tokens) to AFS user IDs (found in directory ACLs)
- Provides file servers with lists of group membership for users to help make decisions about permissions
- Interaction with it is primarily via the *pts* command suite

How Ubik works

- All database servers are known to each other
- One of the DB servers is selected as the *Ubik coordinator* for each database; the DB servers hold an *election* for this purpose
- The election is rigged – all DB servers vote for the DB server with the lowest IP address they can hear
- For a server to be elected coordinator, it must receive votes from at least half of all the cell's DB servers; this is referred to as a *quorum*
- The server that is the DB coordinator is also called the *sync site* for the database

How Ubik works (cont)

- Data can be read from any DB server; data can only be written to the sync site for a given DB. The coordinator then distributes changes to the *secondary* DB servers in approximately real time
- The coordinator sends *guarantee* messages to all the secondary servers every so often asserting that it still has the up-to-date DB for the next 60 seconds
- All secondary servers that hear the coordinator send it *vote* messages saying that they'll trust it for the next 120 seconds
- If a coordinator fails to get quorum at any point, a new election is forced and takes about three minutes
- If an election fails (no DB server gets a quorum of votes), the entire database becomes read-only until a future election succeeds

Update servers

- This type of server has both a server and a client component
- It is used to distribute new AFS binaries and config files to all servers in the cell
- Usually one server of each platform type will run *upserver* and the rest will run *upclient.* Additionally, a single server designated as the System Control Machine will run *upserver* and all other servers will run *upclient*
- New binaries are installed to a machine running *upserver* using the *bos install* command, and the rest of the servers pick them up within a few minutes
- To maintain consistency between the servers, all the servers in a cell are generally restarted simultaneously with new binaries at the most unobtrusive time possible

Basic OverSeer Server

- All AFS servers also run the Basic OverSeer Server (*bosserver)*
- Monitors and controls the rest of the AFS server processes and restarts them if they crash (can also notify sysadmins)
- Stops the *fileserver* and *volserver* when either one crashes and runs the *salvager* to fix any corruption before restarting them
- Manages core files produced by server crashes
- Can restart all server processes at a preset time to avoid leaking too much memory (the athena.mit.edu cell servers do this at 6am every Sunday)
- Invokes new binaries distributed by the Update Server
- Allows sysadmins to examine logs and run arbitrary commands without being directly logged into a server

Cloning

- A *clone* of a volume can be created on the same partition as the volume itself.
- It copies the file structure of the original volume and references the same vnodes, so it takes up almost no space.
- As files are changed and saved in the original volume, they’re assigned new vnodes, thus causing the clone to grow in size.
- Clones are used to create backup and read-only versions of volumes.

Replication

- Volumes can be replicated - a read-only clone of the volume is placed on different fileservers, so it's available even if something bad happens.
- A *.readonly* suffix on a volume name indicates that it's replicated
- Most software volumes are replicated
- When a new version of a read-write volume is released, the Volume Server on the RW volume's file server creates a new release clone volume and distributes it to all the replication sites, while the Volume Location Server keeps track of which sites have the up-to-date version. Clients will only read from the up-to-date version.
- This is what is happening when you see a zephyr like "Releasing user"
- Read-only copies on servers that don't house the read-write copy take up as much space as the read-write copy.

Replication Consequences

- /afs/.sipb.mit.edu (complex rules)
- Requirement that each volume up to root.afs is replicated
- Need to release replicated volumes to commit changes
- Picks which replication site to use using a prioritized list controlled by network distance (between 5000-40000) + afs_randomMod15().

Cache Manager

- Not a single process, like the others
- Initiated by afsd
- Part of the kernel - the only part of AFS that runs on a client machine
- Handles Rx events (Rx is AFS's version of remote procedure calls (RPC). It's what sunrpc is for NFS.
- Caches Data.
- Tracks the state of cached files via callbacks sent by File Server. To indicate that a file has changed, the File Server breaks a callback, and the Cache Manager requests a new copy

How you get a file from AFS

- Client's Cache Manager contacts Volume Location Server to find out which File Servers have the file in question
- If the volume is replicated, and its mountpoint is in a read-only volume, Cache Manager chooses a read-only copy from a File Server (unless the mountpoint specifies the read-write volume). Otherwise, Cache Manager chooses the server with the read-write volume
- Cache Manager talks to File Server and gives it a token if it hasn't already done so
- File Server asks Protection Server for the user's group list if it hasn't already done so, and looks at the directory ACL to decide whether to grant access
- If access is granted, File Server sends the file data
- File Server also provides one callback per file for read-write volumes, and one callback per entire volume for read-only volumes
- File Server will break this callback if the file or read-only volume changes to let the Cache Manager know its data is obsolete

Files in `/usr/vice/etc`

- `CellAlias` - aliases (`/afs/athena/` vs `/afs/athena.mit.edu/`)
- `SuidCells` - "trusted" cells that we allow setuid programs from
- `ThisCell` - the cell you're in
- `CellServDB` - maps cells to database servers - if you're not listed in here, other cells can't see you. Updated 3-4x per year.

afsd options

- fakestat: make “ls /afs” not hang by lying
- dynroot: use CellServDB to populate /afs and don't actually contact cells until access
- afsdb: Do DNS lookups to discover cells
- Also options to control cache manager's number of processes and various parameters of cache size

AFS client startup

- Magic is performed to determine the appropriate module to load (on Linux); module is loaded into kernel
- afsd is started, and the cache is initialized.
- encryption is negotiated (if possible and enabled)
- The cell's root.afs volume is mounted at /afs (but if –dynroot is used, /afs will contain cells from CellServDB instead)

AFS/NFS Translation

- AFS's NFS and DFS translator code is responsible for some of the badness in the source.
- MIT has two AFS/NFS translators, of which it would like to disavow all knowledge: ni.mit.edu and atalanta.mit.edu
- They also support anonymous ftp to obtain files in your Public directory, or any other “system:anyuser rl” directory.

AFS Command Suites

- Command suites are “meta” commands. They do nothing by themselves, but take additional commands and arguments.
- Most commands within the suites have aliases for convenience.
- All command suites understand the “help” command, which lists the available commands within the suite, and “apropos”, which searches the help texts.
  - bos - Controls bosserver
  - fs - Controls Cache Manager/File Server
  - pts - Interface to Protection Server
  - vos - Interface to Volume Server

bos

- You can’t do much without bits, but it’s useful for example.
- `bos status` *`servername.mit.edu`*
- `bos listusers` *`servername.mit.edu`*
  will tell you who the gods are (Susers)

AFS ACLs

- Specified for user or group
- Both positive and negative ACLs; negative takes precedence
- Seven bits (Read, Write, List, Insert, Delete, locK, Admin)
- Eight bonus bits (ABCDEFGH) for local use
- Specified as the letter for each bit, or as one of four words (“read”, “write”, “all”, “none”).
- `fs sa` *`dir {user | group} bits`*
- `fs la` *`dir`* (*dir* defaults to current directory)
- Groups are specifed as owner:groupname (ie: system:athena-rcc)
- Special groups:
  - system:anyuser - anyone
  - system:authuser - anyone with tokens for the cell
  - system:administrators – the gods (implicit la everywhere, etc.)

fs

- You'll use this most frequently. You already know the `setacl` and `listacl` (`sa` and `la`, respectively) commands.
- Other useful ones are (note that most get/list also have a set version)
  - `checks` - check if file servers are up
  - `listquota` - list quota and partition usage in both KB and percentage
  - `quota` - list quota usage in percentage
  - `lsmount, rmmount, mkmount` - control mount points, like OldFiles.
  - `whereis`, and `whichcell` - display the servers (sorted by serverprefs) and cell (respectively) of a given file
  - getcrypt – display whether DES cryptography being used
  - getserverprefs – display server preferences for replicated volumes
  - sysname – get/set the sysname list used for @sys resolution
  - flush/flushvolume – flush the AFS cache

pts

- Useful for examining AFS group information.
- system:groupname and owner:groupname or simply groupname. system: and prefixless groups are administrator-controlled.
- `pts mem` system:group
- `pts exa` user or system:group (or owner:group)
  - `pts exa` outputs flags for control of pts commands on that user/group.
  - Flags are: s (examine), o (listowned), m (membership), a (adduser), r (removeuser).
  - Values are: lowercase letter (members of the group), uppercase letter (everyone), or “-” (system:administrators).
  - At MIT, these are inherited from moira for system: groups. Thus, if a list is visible, it gets a M flag.

vos

- “examine” is likely to be the only one you’ll use. Examine takes a volume name. Forgot your volume name? `fs lq` will tell you, as will `fs lsm`, but `fs lsm` doesn’t take “.” as an argument
- `vos exa user.paco`
- `vos exa sw.matlab`
- `listvol` tells you what volumes are on a given server. Not terribly useful.
- Most other things only admins can do.

Tokens and AFS UIDs

- Tokens authenticate you to an AFS cell
- Really a Kerberos 4 ticket in disguise (hence DES)
- Stored in kernel PAGs (2 fake GIDs between 32k and 48k or default PAG; keyring support in future)
- Can have tokens for more than one cell simultaneously
- User security based on AFS UID (not necessarily the same as UNIX UID)

Token and PAG Manipulation

- Token manipulation:
  - `tokens` - display your tokens
  - `unlog` - kill your tokens
  - `aklog [-force] [-c cell]` - get your tokens, -force forces it to get new tokens even if you have some. (Necessary if group membership changes and you want access immediately)
  - klog - what you'd use if MIT were using the kaserver.
  - groups – shows the fake GIDs, and thus what PAG you're in (and complains!)
  - newpag, pagsh are PAG manipulation tools.

Weird Quirks

- The owner of a volume has implicit la
- No byte-range locking (breaks sqlite v3)
- Full-file locking is only advisory
- close() can fail (!)
- Cross-cell authentication (ex: athena and csail)
- AFS partitions must match “/vicep[a-z][a-z]?”
- Normal fsck will wreck AFS partitions. AFS distributes a special one.
- .__afsXXXX files (deleted, but still open)

MIT Conventions

- fs df /afs/cellname/service/partitions/* lists the partitions and which ones have space
- At MIT, volumes get renamed from user.foo to Xuser.foo when user foo is deactivated.
- Athena's @sys usage (tune in next week)
- Using primarily system:groupname (administrator-controlled) groups maintained by moira
- tabbott and tabbott.root (note krb5->krb4 change of “/” to “.” as delimiter) are distinct AFS identities. MIT assigns additional pts IDs for a given user by adding 64k
- The sipb cell uses the same UIDs as the athena cell (and also the same kerberos realm, which is convenient
- system:expunge (for “delete” command)

Tools

- The following tools may be helpful for debugging AFS: fstrace, cmdebug, rxdebug, udebug.
- Good luck using them. We recommend running $command -help first.
- pt_util (edit prdb directly)
- Sam Hartman's afs-newcell script in Debian if you want to run your own cell for some reason
- There's also an AFS API perl module

References

- www.openafs.org (has extensive AFS documentation)
- grand.central.org - good AFS FAQ
- Arla: a free AFS client - started before OpenAFS (supports FreeBSD)
  - www.stacken.kth.se/projekt/arla/

Acknowledgements

- Thanks to Jonathan Reed for letting us steal his OLC slides as a starting point
- Thanks to Derrick Brashear for answering questions about early AFS history