

mailstore - Explaining how mails are stored on disk - the big picture

Carlo Contavalli

ccontavalli@masobit.net

Revision History

Revision 1.0.0 2006/02/26
First document revision

This document will try to document how emails and user data is written on the disk by PigeonDeliver, and how the various modules interact in this writing process.

1. Introduction

The only and official way for PigeonDeliver to store emails on the local hard drive is by using the mailStore service.

Purpose of the mailStore service is to save all the emails being received somewhere on your hard drive, in the administrator/user specified format.

The mailStore service is modular in nature: depending on the modules the administrator will choose, users home directories might be created automatically, lazy allocation routines might be used, or users/admins might have a greater degree of freedom by choosing which format to use for the mailboxes or how to create the directory hierarchy itself.

Talking about the structure, the mailStore service is quite simple. It uses two kind of modules:

mda

which takes care of writing the mail in the mailboxes, mailboxes that can range from simple directories on the disk to mailbox files, database or anything else you may like...

creator

which mainly takes care of handling concurrency when creating new users (more details will be given later).

The main idea behind this structure was to make sure that the user database was kept as simple as possible: when a new user is created, how is a web interface supposed to know which home directory to use for the user? How is it supposed to know which uid to use? What happens in a clustered environment? What happens if those information get out of sync for any reason? What happens if the administrator wants to change the directory structure or the on-disk format for mailboxes?

When a new user is created, and when "mailstore" (mail saving on disk) for this user is enabled, a simple record is written into the user database (either by the administrator, tools or web interfaces). This simple record just needs to tell the name of the user and all those configurations wich the administrator wants to be forced on the user itself (usually, user quotas, if a paraticular directory needs to be used, and so on...).

When a new mail is received, the mailStore is called. The mailStore itself parses the provided user configuration using the MDA module specified in the configuration file.

The MDA module will tell if the provided information is "enough" to deliver the email, or if some additional information must be provided.

If nothing is missing, the MDA module will just be called to perform delivery.

If something is missing, the mailStore will take care to call the "creator" module. The "creator" module itself will call the MDA once again, asking it to allocate the user, which means, to provide all the "missing data". If everything works, the "creator" will then add that data in the user structure on the database. So, in short:

- the MDA is able to parse user configurations and verify if all the necessary informations are there, which often means that it is able to tell if the user "has been created on disk" or not. If it has not, the MDA is able to "allocate the user", which means it is able to provide all the missing data, which often means creating some directories on the disk or writing some files on the system.
- the creator itself is able to write all that additional data into the user database. But, hey, what happens in a clustered environment if another server has been faster than us in allocating the user and adding his own data into the database? what happens if another processe receives the same mail and allocates the mail even if we already started the allocation? The main purpose of the creator module is to handle concurrency: some creators might just lock the database while the user is created, others may use a test-and-trial procedure, some others may just ignore the problem.

Keep in mind, however, that what exactly is done when a mail is received or when a user is allocated just depends on the MDA being used. For example, a database based MDA may create a new "table" when a user is allocated, and then write each mail as a record into that table. A file system based MDA may just create the user home directory upon "allocation", and write the mail in a mailbox file whenever the delivery has to be processed. Right now, there are not that many MDA modules, but we are planning to write many of them with different features. It is fairly easy to add a new on-disk data storage by simply writing one more MDA module.

2. Creator modules

At time of writing (Sun Feb 26 12:36:59 CET 2006), there are mainly two creator modules:

simple

which simply ignores any kind of concurrency issue. It should be used only when you are sure no more than one delivery at a time will be performed, or when some other PigeonDeliver module is taking care of handling concurrency.

concurrent

which uses some kind of lazy approach: it assumes that most of the times concurrency issues will not arise, and that "allocating" and "freeing" an user are fairly simple and reversible operations. That said, it simply tries to "allocate" the user. When adding the user to the database, it ensures that no data can ever be overwritten. If an "overwrite" error verifies, the user is "freed" (the "allocate" operation is undone), and the module tries to fetch the new data from the database for a configurable number of times, waiting between each fetching a configurable amount of time (this is important when the database is replicated, and the replication times cannot be ignored).

Keep in mind that the creator is called only when a new user receives his first email, or when he first logs in (which are very rare events compared to regular users logging in/out and to emails being created). Whenever it is called, however, it is very efficient on most cases. On the very few cases where a conflict is to be handled, which is very unlikely to happen even on very crowded systems, there may be a small delay in delivering the mail/creating the user, delay that is unnoticeable by end users. It doesn't also impose any additional load on the system, behind the load additional load of a useless "alloc" and "free" operation.

The "delay" can be tuned, configured, or even zeroed. It is used mainly to allow replicated or distributed database to propagate the changes to the database. So, if you are using one single database server, you shouldn't worry about this.

Other and new creator modules may easily be created. In one of the next PigeonDeliver releases, you will probably see a new "lock" creator, which creates users by locking the database (record locking whenever possible) and thus avoids any possible concurrency issue. We expect this module to have much worse performance than the standard "concurrent" module, unless an MDA where the cost of "allocating" and

"freeing" an user is greater than the cost of "locking", "unlocking", and possibly blocking some of the other operations on the database every time a new user is added.

Keep also in mind that for most MDA modules, allocating a user just means creating a directory on the local file system (one mkdir call) or just picking a random server to be used.

3. MDA modules

Well, MDA modules are actually those modules able to write emails. It is the MDA module that actually decides if to write the mail on the file system, where on the file system, to write the mail on a database, to send it over to another host using some other protocol or so on...

There are actually two kind of modules: local filesystem based and non local or non filesystem based. There is no real difference between them and there is no real way to distinguish the two kind of modules. However, while non local or non file system based modules are all very different from each other, most of local filesystem based modules look alike. Internally, they all:

- need to handle/create/... a directory, or a place where all the mails will be stored, eventually setting and handling user privileges.
- need to store emails in these directories one way or another, and write emails/folders on the file system.

Even database based local filesystem modules need a directory somewhere to be used, and to write emails in some files that may be local to the user or organized in a directory hierarchy.

Many MDA modules are thus actually split-up into two smaller components:

- the SDA, or Simple Delivery Agent, which is just a trivial module able to write emails on the disk. It assumes it has a writeable directory where to store data, and that it can do whatever it likes in that directory to store the email on disk.
- the MDA itself, which needs to create the directory to provide to the SDA, handle privileges, uids, gids and generally make sure everything works ok.

Examples of SDA may be the "maildir" module or the "mailbox" module, the first one able to create a "maildir" in the directory specified by the MDA, the second able to write a plain mailbox file in that same directory. Examples of MDA can instead be the "dscmtree" module, which organizes users in order to have each one its own directory named after his own uid or gid, or the "systemtree", which organizes folders based on their own home directory in the passwd file.

Note also that while most MDA use one same UID/GID to deliver emails to users, the general PigeonDeliver approach is to have each user with its own uid/gid. *No, this doesn't mean you have to add*

all of your users in the passwd file. The passwd file is just used by some tools to map uid/gid numbers (stored in the filesystem, used by processes) to user names. You can use an uid/gid even if the name of the user is not in the passwd file. If you don't like this, you can always use nsswitch.conf and read uid/username mappings from the database of your choice.

If you don't like having each user with its own uid/gid, or your system still has only 16 bits for uids, limiting the system to at most 64K users, you can always configure the various modules to use one single uid/gid to handle permissions.

In the next few sections, we will talk about some of the main MDA modules available for pigeondeliver.

3.1. dscmtree

dscmtree is the default PigeonDeliver MDA agent. It stores mails on the local file system. If the default is not changed from the configuration file, in order to use this module you will need to create a directory named /dscm/data, where all of your users will be put. To change the name of this directory, you can use the "mailStore_base" configuration parameter.

In this directory, an hierarchy based on server name and user id is created. The default server name is "local". To set a server name, use the parameter "mailStore_server". For example, if you are using the "dscmtree" with the default parameters and you correctly created the /dscm/data directory, after the first mail is received, you will see something like:

```
$ cd /dscm/data
$ ls
local
$ cd local
$ ls
00 07 0E 15 1C 23 2A 31 38 ..
01 08 0F 16 1D 24 2B 32 39 ..
02 09 10 17 1E 25 2C 33 3A ..
03 0A 11 18 1F 26 2D 34 3B ..
04 0B 12 19 20 27 2E 35 3C ..
05 0C 13 1A 21 28 2F 36 3D ..
06 0D 14 1B 22 29 30 37 3E ..
```

don't panic! It's all right... Every time a mail is received, the "dscmtree" MDA module is called to deliver the mail. The dscmtree module checks the user configuration. If the user has a mailuid assigned, and if his home directory (read below!) seems to exist, the user is assumed to have already been created.

In any other case, the "creator" module will call the MDA asking for it to "allocate" the user. The "dscmtree" module simply assigns a "mailuid" to every mail user of the system. This mailuid is used as the name of the home directory of the user himself, and, if privilege handling is not disabled, as the uid to

be used to perform the delivery. mailuids are always assigned from an administrator specified range, which defaults to 1000 - 65531 on systems with 16bit uids, while to 1000 - 500000 on systems with 32bits uids.

So, when a mail for ccontavalli@masobit.net is first received and needs to be stored on the server, a dscmtree with the default configuration will first assign a mailuid to the user, and then create his own home directory. mailuid are always 32 bits long, and the home directory will look something like: /dscm/data/local/05/000B89/, where 000B8905 (least significant two digits first) is just the mailuid of the user, in this case 755973. So, the mailuid 755973 is assigned to the user. A corresponding directory has been created, the delivery process has switched to the privileges of that user and the mail has finally been delivered.

The dscmtree module is very tunable: you can easily specify the range of mailuids to use (minuid, maxuid), you can tell to always perform deliveries with the same uid (defuid), to use group uid or the same gid for everyone (defgid), which root to use (base, /dscm/data), whenever to trust any eventual gid already written in user configurations (strictgid) or uid (strictuid) or if they should be verified against the specified range, if to trust the uid to correspond to an already created directory (looseid set to false) and if the uid specified in the database must be mandatory. Obviously, some uid/gid are never accepted for security reasons.

3.1.1. Razionale

Purpose of the whole mechanism is to:

- Allow users to be easily renamed without accessing the filesystem or without losing emails or having to recreate the mailbox. With an uid based mechanism, users can be easily renamed without troubles.
- Avoid a directory hierarchy based on domains and usernames. We wanted the ability to distribute users among a cluster of machines independently from their own domain, to move the users from one domain to another without having to delete the mailbox and to support single domains with huge number of accounts.
- be simple, fast and reliable, even on clustered environments. Using the file system as the only source of information for usable uids means that a consistent view will always be maintained, no matter what happens. No locking, communication between machines or similar mechanisms are needed to keep everything up and running, even when there is a crash on one of the nodes or some data is lost.
- Use file system and OS facility as much as possible. We didn't like to use a single UID to perform deliveries. We wanted the daemons (IMAP/POP3/Delivery) to have at most the privileges granted to the user they were acting upon, and not more.

When enabled, using the mailuid as the system UID has at least three great advantages:

- file system quotas can be used.
- the daemons run with the privilege of the users.
- privileges, ACL, and so on are enforced by your OS kernel.

- Keep the web interfaces, the database and the administrative commands as simple as possible. Most web interfaces out there calculate an uid to be used for the user, write the home directory in the database, and all kind of stuff related to the system and to the kernel.

Those ideas simply don't fit in a clustered environment: a web interface shouldn't decide the home directory of a user and shouldn't calculate an uid for them, as much as it shouldn't access the file system in any way and shouldn't perform any change outside the database.

When somebody writes a VB application or a Perl script to handle users, he shouldn't worry about choosing an uid or in which directory to write his own data or how to handle files and directory when the user is renamed or moved to another domain.

The final result is that you can easily switch from one mechanism to another, add nodes to a cluster without troubles and generally be sure that things will keep working without troubles as time goes on.

3.1.2. UID allocation algorithm

In order to calculate an UID to be used for the user, the "dscmtree" module simply gets the name of the user and calculates a hash from that name. The hash is mapped in the range of the allowed uids (minuid and maxuid parameters), and the corresponding directory is created.

If the creation succeeds, the UID is valid and used. If it does not, the algorithm uses a quadrating hashing technique to calculate a new hash, and the process starts over.

After quadlimit attempts, the algorithm is switched to a linear search to find the first available uid. If it is found, that uid is used. Otherwise, the "dscmtree" module fails since no more uids are available. In this case, you should increase the range of available uids by tuning the minuid and maxuid parameters.

On average, the algorithm is really fast. It is very reliable, since it will always find an uid to be used as long as there are uids available, and it won't go out of sync with the content of the file system, even in case of disastrous events.

3.1.3. Quirks about using sparse 32bits uids

The "dscmtree" module chooses pseudo random uids to be used in the range specified by the "minuid" and "maxuid" parameters.

Most modern *nix systems support 32 bits uids without any trouble. Just keep in mind:

- if your system only supports 16 bits uids and you want to have more than 60000 users on your system, you better switch to "single uid" for all users or switch operating system.
- if you are planning on using file system quota, please verify the on-disk format used by your OS to store quota information on the disk. As an example, old versions of linux (< 2.4, I believe) using quota v1.0 use a plain binary file where the user uid is used as some sort of offset in the quota file. If you specify a range from 1000 to $1 \cdot 10^9$, and the algorithm picks up an uid close to 10^9 , your quota file will probably use a couple hundreds gigabytes on your disk (yes, GB!).

Whatch out that the a format very close to linux v1.0 quota files is being used by other BSDish systems.

At time of writing, the filesystem with better support for quota we could try is XFS.

- if you want your system to scale well and smoothly, keep the range of uids usable at least 20-30% bigger compared to the number of users you have. It does not hurt performance in any way to have even a bigger range. So, if you are planning on having at most 200000 users, use a minuid and maxuid that will leave about 250000 uids usable.

3.1.4. Security risks of this mechanism

/dscm and /dscm/data must not be writeable by anyone else beshide the mail system. Never disable the strictuid or strictgid options, even when porting users from an old system.

Those options tell PigeonDeliver to trust the values of the uid written into the database, without performing any check on the range of the uid. Unless you are really sure about what you are doing, do not use them.

3.2. sharedfolder

Shared folder is probably the simplest MDA available: it simply reads a directory name from the configuration file, and stores the emails in that directory (directory). The directory must exist before this module is ever used.