News Drowning:

an alternative method to news expiring or "how not to be drown by news"

Pedro Vasconcelos Rogério Reis

Technical Report Series: DCC-97-13

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

News Drowning:
an alternative method to news expiring
or
"how not to be drown by news"

Pedro Vasconcelos   Rogério Reis
LIACC, Universidade do Porto
R.Campo Alegre 823, 4150 Porto, Portugal
{pbv,rvr}@ncc.up.pt

Abstract

A Usenet `news` server tends to be one of the most resource consuming services in a computer site. Disk-space is specially critical, requiring the most rigid discipline, purging old articles to make space for new ones; this is usually done by periodically expiring articles after a certain age.

We describe an alternative method, that is triggered by the need of disk resources and chooses the articles to purge by attributing a weight to each article, according to its age and newsgroup.

This new method achieves both a better use of available disk-space, and a better response in critical situations avoiding the tedious need of the `news` administrator continuous surveillance.

1 Motivation

A Usenet `news` server tends to be one of the most resource consuming services in a computer site. As new articles are arriving, Usenet `news` maintenance requires periodic purging of old articles to make space for the new ones. With Internet-News (INN) `news`-management software, this is handled by a separate program, `expire` and the process is called "expiring". Typically, `expire` runs automatically (among other `news` activities) on a daily basis (*i.e.*, scheduled by `cron`).

Expiring is parameterized by the number of days to keep articles on each newsgroup (or a whole branch of the newsgroups hierarchy); articles that are too old are removed. This implies that a `news` administrator has to do some trial-and-error attempts to guess the correct parameters for the intended expiring policy using the available disk-space:

- If articles are not expired at the same rate they arrive, the system will overflow. In some cases only human intervention, removing part of the articles, can unblock the situation because the lack of disk-space creates a deadlock and does not allow the work of `expire`.
- If expiring rate is kept too high, the number of living articles will be lower, disk-space will be under-used, and users will have a shorter-term `news` service and thus, less to read.

To avoid the first situation, because of its very annoying consequences, one tends to fall into the second, which is not so catastrophic, typically using little more than half of the disk capacity. This is why an optimal balance is hard to reach, specially considering the "flooding" nature of `news` distribution, and its frequent "wave" behavior:

- occasionally, incoming `news` traffic might increase, in which case articles should be kept less time;
- a `news` server that is feeding us might be in-operational for some time, thus less articles arrive during that period; in this case existing articles can be kept a little longer.

The previous considerations led us to design a `news`-purging method that is triggered directly by the available disk resources and *not* by a fixed schedule table, allowing a more frequent action when it is needed, and a more relaxed attitude when it is not critical; we called this new method "Drowning".

2 Desired functionality

Drowning is based on these simple ideas:

- we should establish a priority among all current articles as to which ones should be purged first — *i.e.*, we must establish a *queuing order* to purge articles.
- we should start purging articles when the available disk resources (either disk-space or *i-nodes*) are below a low-threshold.
- articles should be removed (according to the purging order) until a high-threshold value of disk resources is obtained again.
- the downtime for the `news` server (the period in which it is busy removing articles and doesn't accept new postings) should be kept as low as possible, not only to avoid queuing our incoming feed, but (most important) to allow users to post new articles.

Unlike `expire`, drowning can occur at any time — even during the day, while the users are accessing the `news` service — so the last point is particularly critical.

Note that in order to avoid several intermittent drowning actions the low and high thresholds should be different; in fact, the gap between them and the current incoming `news` traffic will dictate how often do drownings occur.

To establish priorities we attribute each article an *weight* integer value that should reflect its age and relative importance, with higher weights corresponding to older/less important articles. We can then sort all articles according to weights, so that we can remove "heavier" articles first (they "drown" faster).

2.1 Article queuing

The weight of a given article should clearly depend on two things: the article's age (elapsed time since it arrived at our server) and the newsgroup (or newsgroup hierarchy) the article was posted to[1].

Clearly, weight should increase as age increases; also, some newsgroups' weight should increase faster than others (say, the weight of articles posted to "`alt.whatever.*`" increases twice as fast as those on "`comp.something.*`" for the same age) to allow for a site-specific priority among newsgroups.

However, a simple linear function on the age hasn't got the required expressiveness for two reasons:

- novelty is a value in itself (so recently arrived articles should be purged only in last-resort, no matter what our newsgroups priorities are);
- if we re-distribute `news`, we must keep *all* articles long enough to allow them to be sent.

We propose a two-step linear function with two parameters: a "lifetime" threshold a, and a drowning "slope" m, so that the weight function for an article with age t (measured, say, in days) is:

$$w_{a,m}(t) = \begin{cases} t & \text{for } t \leq a \\ m(t-a)+a & \text{for } t > a \end{cases}$$

The parameters a (threshold) and m (slope) can be specified by newsgroup hierarchies (*i.e.*, using wild-card-style patterns) in a "drowning policy" table (see figure 1).

[1] The issue of cross-postings is dealt with later on.

pattern	threshold (days)	slope
`comp.*`	7	1
`alt.*`	4	3
`*`	2	3

Figure 1: Example drowning policy

2.2 Cross-postings

In the previous discussion we ignored the fact than an article is often posted to more than one newsgroup at a time. This is called *cross-posting*.

The INN **news**-server handles this by keeping one file copy of the article, and having several links[2] to this copy, one per each newsgroup it was posted to.

This implies that the article space will only be reclaimed when all the links to an article are removed[3] *i.e.*, when the article is removed from *all* the newsgroups it was posted to.

This remark justifies the extension of the weight function for cross-posted articles: it should be the *maximum* of the weights for each newsgroup the article was cross-posted to.

This definition means cross-posted articles are given the priority of the "*worst*" newsgroup hierarchy in their cross-post list — which has the advantage of desencouraging wide-spread cross-posting.

2.3 Expire headers

The author of a **news** article can choose to express its validity period by putting that information in an article header — called the *expire header*. Because this information is much more relevant to this particular article than the information in the general article expire policy, it should be considered before anything else.

Notice that the expire header can not be used to extend the lifespan of an article — the **news** adminstrator has full authortity to remove an article *before* its expire header date. However, if the expire header date is past then we can remove this article regardless of its newsgroup(s) and age.

We need to extend the drowning algorithm to use this header information to make sure that articles with a past expire header date don't use up more disk-space.

It suffices to attribute an "infinite" weight[4] to articles with a past expire date; this will make sure these articl es get deleted first-thing. Of course, if the expire date is in the future, then the article should be weighted as before.

3 Implementation Issues

Our implementation was integrated with the INN **news**-management software, since this is what we use on both our **news** servers; morever, INN is becoming the standard **news**-management software in the UNIX world.

We observe the following:

1. To establish priority among articles we only need information contained in the **news** history database file (both time of arrival and newsgroups list is maintained there).

2. In order to purge articles according to their priorities, it is convenient to keep a *remove queue file*, used in a FIFO-like structure (*i.e.*, articles are removed following the order in the queue file). Also note that this file can just keep *references* (*i.e.*, offsets) to the history database entries, *not* any actual article information.

[2] Links are either *hard-links* if they're in the same file-system, or *soft-links* if they're in a different file-system.
[3] This is the normal semantics in UNIX of unlinking a file with "hard-links" pointing to it.
[4] In the implementation, we will attribute these articles the largest positive integer representable on a machine word.

3. It is more convenient to remove "batches" of articles to reduce the granularity of disk accesses[5].

4. We need to remove the article's files in the `news` spool and also to modify the corresponding history entries (indicating that the articles were deleted). To see how this is done we must look into the history database in greater detail.

3.1 Purpose of the history database

The `news` history file contains a line for every article in the `news` spool, with the following information:

Message ID: a string uniquely identifying the article;

Arrival/Expire & Post times: the dates of arrival/expiring/posting for the article (not all fields need be present);

Newsgroups & Files: the list of newsgroups the article was posted to and corresponding file-names within the newsgroups spool area

The history file maintains not only information on every article that is still in the `news` spool (*active articles*) but, for a period of time, also remembers information on articles that were removed (*remember entries*). The reason for keeping remember entries is to prevent getting the same article twice (should some `news` feeder offer us again an old article). After a certain number of days, the remembered entry is finally discarded.

The remember entries in the history file contain just the "Message ID" and posting time fields — no newsgroups and files since they're not in the `news` spool anymore.

Note that this has *no influence whatsoever on which articles are removed* or when are they removed: some articles may be removed before the remember time, in which case, we keep a remember entry, while other may be kept longer, in which case, we need not. Only the drowning policy and available disk-space dictate what is kept or not.

3.2 Updating the history database

To facilitate lookup, the history file it's is kept as a `dbz` database: a text file with an external hash-table which associates each article identifier (message-id) with an offset into the text file. This means that any update to the text file must also coherently update the `dbz` database.

Deletion of entries, in particular, implies a complete rebuilding of the database (because of the changes in the offsets), and is carried out by first creating a temporary new database and then overwriting the old database with the new one.

Fortunately, as we shall see, this costly task can be performed on a specified schedule (say, during the night), and is not required after each drowning task.

4 Implementation

Drowning was implemented as a collection of separate UNIX processes that communicate through files and pipes. Some `Tcl` shell-scripts coordinate these processes.

UNIX standard utilities (such as `sort`, `cut`, `mv`, `rm`) are also used to process intermediate files, as well as INN-specific utilities (`ctlinnd`, `fastrm`, `expireover`).

We found that this modular approach gave us a working prototype in a very short time, as well as making it simpler to experiment new approaches and tailor to specific needs; however, to guarantee efficiency, the "inner cycles" of computation, such as processing the large `history` file, are written as C programs.

[5]Depending on the average article size (say, 2–10Kb) typical batch sizes should be within 1000–20000 articles.

4.1 Files & scripts

Apart from the binaries and scripts, which all go into the standard `news` binaries directory (say, `/usr/lib/news/bin`), some extra files are used:

`drowning.ctl` is a text file specifying several running options including the drowning policy (see section 5); it goes into the `news` standard directory (say, `/usr/lib/news`).

`history.que` is the remove queue file. It contains a list of offsets to entries in the `history` file, ordered ascending in article priority (see section 2.1).

It is created automatically from `drowning.ctl` and the `history` file; it goes into the `news` standard directory (say, `/usr/lib/news`).

`drowning.log` is a log-file that records actions done by any of the drowning utilities. It goes into the `news` logs directory (say, `/var/log/news`).

4.2 drownwatch

`Drownwatch` is a "watch-dog" script that should be run in the background and periodically pools available space and *i-nodes* in the relevant file-system. If needed, it will start a drowning action to remove articles, locking access and throttling the INN daemon as required, building up reports, etc.

The drowning action itself consists of invoquing some commands in sequence:

`drbatch` reads a "batch" of article offsets starting from the end of the `history.que` file and produces a file with path-names[6] corresponding to the files to be removed; the batch-file will later be passed over to `fastrm` which actually unlinks the files.

Also, `drbatch` converts each article to be deleted into a remember entry in the history file (see section 3.1) by overwriting part of its entry with padding spaces[7], This maintains entry size constant and thus entries offsets — ensuring that `dbz` information is still valid.

Finally, `drbatch` truncates the `history.que` file, removing the entries for the removed batch.

`fastrm` receives the batch-file with path-names and actually unlinks the files[8].

`expireover` can be used to update `news` overview information (if the server is using it).

Note that several runs of `drbatch` and `fastrm` may be needed to bring disk resources to acceptable values (depending on the batch size and on the actual resources taken up by the articles). Successive runs of `drbatch` "consume" the the `history.que` file from bottom-up — hence the analogy to article "drowning"!

The only critical part of `drownwatch` that requires locking access to the `news` server (thus limiting users and news-feeds from posting new articles) is the building of the remove batch by `drbatch`. After this the actual removing of the articles can be performed concurrently with normal server activity. Since `drbatch` runs for short periods of time (tipically under 1 minute) users may not even notice the drowning actions taking place.

4.3 drowndaily

`Drowndaily` runs periodically to clean-up the `history` file. It should be run independently of `drownwatch`, typically once a day. It performs several tasks invoking other utilities:

- `drforget` rebuilds the history, eliminating remember entries that are too old (see section 3.1)[9].

  This also eliminates the "padding" spaces that `drownwatch` introduces when it removes articles.

[6] By looking-up the `history` file; see section 3.1.

[7] The remember entry is always shorter than the corresponding active entry, thus padding is always possible.

[8] Better still: sort the batch-file lexicographically first then use `fastrm`, which optimizes the locality of disk accesses for unlinking from the `news` directories.

[9] The remembering time is a parameter to `drowndaily`, as was to `expire`.

- `drbuild` rebuilds the `history.que` file and computes new weights for all articles. The queue file is then sorted by weights.

In order to rebuild the `history` database, `drowndaily` needs to create a temporary database and then overwrite the old one, thus (roughly) duplicating the space taken by history database. If it is run when there is not enough space then it exits, marking that `drowndaily` should be run after the next `drownwatch` that actually frees space. This assures that even with large history databases, there is enough space when history rebuilding is done without requiring that a large space is *always* available.

Notice also that between runs of `drowndaily` the `history.que` can get shorter (due to runs of `drownwatch`), but can also grow due to the incoming feed (see the next item, `drownqueue`). However, it is only `drowndaily` that re-creates `history.que` by weights, thus implementing the drowning policy.

`Drowndaily` also builds a report file that can be sent by e-mail to news administrator; it displays statistics for each newsgroup hierarchy specified on the drowning policy including total articles posted as well as oldest, average and newest article time (in days). This report helps news administrator fine-tune the drowning policy and get a general view of how the news service is maintained.

4.4 `drownqueue`

`Drownqueue` is a "dummy" news feed-queue program that appends an entry to the `history.que` for every article fed to us.

This garantees that we don't run out of the `history.que` file even if we don't execute `drowndaily` at all; of course, this doesn't implement any drowning policy at all (newer arrived articles are simply appended in order of arrival), but we should expect to run `drowndaily` before we ever need to remove these articles.

5 Control file syntax

The `drowning.ctl` is a text file specifying several running options for Drowning (including the drowning policy).

Lines beginning with `#` are comments that extend until the end of the line. Extra white-space (including new-lines) is ignored.

Declaration start with a keyword and a variable number of options. Valid keywords are listed bellow:

remember *days*: specifies number of days to maintain removed entries in the history database;

lo_space *kb*: specifies the value of free space (in Kbytes) that triggers Drowning.

hi_space *kb*: specifies the value of free space (in Kbytes) that Drowning tries to obtain.

lo_inodes *i-nodes*: specifies the number of i-nodes that triggers Drowning.

hi_inodes *i-nodes*: specifies the number of i-nodes that Drowning tries to obtain.

batch_size *articles*: specifies the number of articles in a batch (that are processed at a time)

batch_max *n*: specifies max. number of batches (*i.e.*, runs) before giving out (to prevent looping)

overview: signal that the server is using `news`-overview (NOV) (default is not to use `news`-overview)

newsmaster *addr*: e-mail address of whom to send the drowning reports

verbose *n*: verbosity level for reports (from 0 to 2)

drowning $\{\{pat_i\ thres_i\ slope_i\}^*\}$**:** table of patterns, thresholds and slopes that specify the drowning policy.

6 Further work

6.1 Split file-systems `news` spool

When a large amount of articles are to be kept, the **`news`** spool can be split by more than one file-system. Different hierarchy branches will then reside in different file-systems.

Drowning does not currently support split file-systems spools. To do so:

- each file-system should be checked for free space and *i-nodes* independently of the others;
- the building of a remove batch must take the article's posting file-systems into account, since it is useless to remove files from one file-system when only another is needing space.

The last point is particularly important since it means that a single queue file (`history.que`) is inadequate for split-file systems; it would be better to maintain several queue files, one for each file system. Drowning could be extended to support this.

On the other hand, current UNIX operating systems[10] support "spawned" or "stripped" virtual disk devices that are mapped to several physical disk partitions or devices; some operating systems (Digital UNIX, for example) even allow incremental addition of devices. Also, hard disks are growing in sizes even faster than news spool space requirements, and large 6–9Gb disks are becoming common and comparatively inexpensive.

On the whole, we think that the need for split file-systems may be less important for a small to medium news server than it once was in the past.

7 Conclusion

We use drowning in two of our news servers for almost two years now with good results. These servers have diferents roles (one is exclusively used for news reading, while the other is a news feeder to 6 other sites).

Both servers are PCs running Linux and maintain a full news feed; their news spool space has been updated from an initial 3Gb to the current 8Gb, and drowning has been able to maintain a good use of the available space in different configurations.

Our news policy is to give higher priority to the hierarquies for `comp.*`,`gnu.*`,`sci.*`, ... ; lower in the priority cames `soc.*`,`rec.*`,`misc.*` and lower still `alt.*`; finally anything else (usually newsgroups we are feed by mistake) receives even lower priority.

The table bellow illustrates a `drowndaily` report at the time of writing for the news reading server (times for article lives are in days):

#posts	oldest	avg.	newest	newsgroups
64883	7	4.2	0	comp.*
382	7	4.3	0	gnu.*
2642	7	4.3	0	pt.*
6586	7	4.3	0	news.*
133542	7	4.3	0	rec.*
10127	7	4.3	0	sci.*
42815	7	3.6	0	misc.*
114124	5	2.2	0	alt.*
19961	6	3.5	0	soc.*
3603	6	3.0	0	talk.*
499	5	2.8	0	bionet.*
3824	5	2.5	0	bit.*
71	5	3.2	0	eunet.*
5923	5	3.6	0	biz.*
0	-	-	-	control.*
18772	1	0.4	0	*

[10] Such as the Linux free-UNIX system for PCs, as well as most other modern UNIX implementations.

We point out some remarks:

- Like any news-spool administration scheme, drowning requires a minimal spool space to work efficiently; if too little space is available altogether, then the system "thrashes" (*i.e.*, spends more time removing articles than serving news).

  The wise solution is, of course, to limit the incoming feed rather than rely on drowning (or any other method for that mater) to remove articles.

- If the available spool space is not too small but is nonetheless constrained, the slope parameters do not introduce differentiation among newsgroup hierarquies (only the threshold does) because articles are kept for a minimal time and don't "age" in the server.

  If, however, the available disk space is increased the slopes begin to distinguish groups, as it was originally intended by the news administrator.

Our last remark should of course be that we feel drowning freed us from the tedious task of monitoring overflows in news-spool space.

And even full-time news administrators will enjoy the time do other things in life than watching Usenet news articles drown by ...