The following paper was originally published in the
Proceedings of the Eleventh Systems Administration Conference (LISA '97)
San Diego, California, October 1997

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

The Cyclic News Filesystem: Getting INN To Do More With Less

Scott Lystig Fritchie – Minnesota Regional Network

ABSTRACT

When Usenet News servers were first implemented, the design principle of storing each Usenet article in a separate file appeared to be sound. However, the number of Usenet News articles posted per day has grown phenomenally in the past decade and shows no sign of abating. To stay ahead of the growth curve, Usenet administrators have been forced to buy faster machines, more RAM, and many more disk drives. Many of the performance limitations are caused by interactions with the underlying OS's filesystem, which is usually a Berkeley Fast Filesystem (FFS) derivative.

The Cyclic News Filesystem (CNFS) was designed to avoid most of FFS's major problems when used with INN: synchronous file linking/unlinking and sequential scanning of directory files. Articles are stored within a relative handful of large files, either as regular files on top of a standard filesystem or as block disk devices. Articles are stored sequentially within each file, resuming at the beginning of the file when the end is reached. Disk activity is reduced by an order of magnitude.

Introduction

Though Usenet server software packages have changed greatly over the years, almost all have one implementation detail in common: the method used to store articles on disk. The newsgroup hierarchy maps directly onto a filesystem directory hierarchy, and storing articles in individual files fits the paradigm nicely.

But Usenet article volumes have grown exponentially for at least ten years. Confirmation can be found by reading [Collyer], [Salz], [Swartz93] (whose exponential forecast in 1993 underestimates current 1997 volumes by a factor of four), and by asking any seasoned Usenet system administrator. If that same administrator were asked, ''Does your Usenet server have sufficient capacity for current article volume?'' the answer would probably be ''No'' or ''Barely.'' Nothing appears to be on the immediate horizon to slow down article growth rates, which means most Usenet servers will have severe performance problems soon, if not already.

The ''one file per article'' storage method continues to be used by INN, the most commonly-used Usenet server software in the world today. Most of the operating systems running underneath INN also use a variation of the Berkeley Fast Filesystem (FFS). The FFS, while a significant improvement over its predecessors [McKusick], interacts quite poorly with INN. At today's volumes, an INN server may store thousands to tens of thousands of article files within a single directory. FFS performance degrades quickly when managing large directories, due to sequential directory data scans and synchronous directory updates.

One avenue for addressing this pressing problem is to alter or abandon the current method of storing articles. This paper discusses both. The first stores articles in individual files that are named by an MD5 checksum of their Message-IDs. The other stores thousands of articles within a single file, utilizing the file as a huge cyclic buffer. These buffers can be large files on top of a standard filesystem (even FFS-based) or a block disk device, bypassing standard filesystems entirely. A side benefit of this method is that there is no explicit article expiration process: articles are automatically overwritten as new articles arrive. The resulting performance, as measured by article throughput, increases by a factor of three to four; disk I/O is reduced a factor of 10. There are many nontechnical reasons why the cyclic buffer storage method is preferable to the current method. Most of them are expressed in [StSauver]:

> My throughput is up, my news spool disk I/O bottlenecks have disappeared, I no longer am offline for longer expiration-related periods, I don't need to screw around with software disk striping utilities, and I don't run out of space in /var/spool/news. It's great, and it's free.

The rest of this paper is organized as follows: the next section follows an article through INN's processing and describes how an FFS-based filesystem adds a significant hidden cost to that processing. Then, the INN Improvements section describes previous steps taken to lower that cost. The next two sections describe two of the author's methods to change INN's article storage method to lower or avoid FFS overhead. The Performance section compares the performance of a standard INN server to one utilizing the cyclic storage method. The final sections provide some directions for future INN-related research and some concluding observations.

Life As an Article Processed by INN

INN has separate software components that process an article during its four phases of existence in an INN server. Each phase has problematic interactions between INN and FFS. They are as follows:

1. Receive the article and store it in a file in the spool directory.
2. Read the article file from the spool directory to forward it to peer servers.
3. Read the article file from the spool directory to forward it to NNTP reader clients.
4. Remove the article file from the spool directory.

An Article Is Received

INND is the central daemon process that accepts or refuses articles offered via the NNTP protocol [Kantor], stores the article, keeps a database of articles it has already processed (in `history` and related DBZ index files `history.pag` and `history.dir`), and decides which peer(s) the article should be forwarded to.

See Figure 1 for an example article that INND has just accepted. INND consults the `active` file to assign this article the next available number for "misc.jobs.offered", which could be 9124816, and stores the article in a file with the path `misc/jobs/offered/9124816` (relative to the SPOOLDIR directory, e.g., `/var/spool/news`). When storing the sample article, the open() system call will trigger a behind-the-scenes scan of several directories first. (See Figure 2.)

The directory name cache may eliminate the need to fully scan directories 1-6, and the file buffer cache may avoid the need for some or all of the disk I/O. Articles tend to arrive with a fairly random newsgroup distribution, which makes the OS caches less effective without large amounts of RAM to assist them. In the worst case, directories 1-6 must be scanned sequentially to find the inode for directory 7, reading 194KB of directory data; each scan would trigger at least one disk I/O.

Once the inode for directory 7 is found, it must be read sequentially to look for an unused space to insert file `9124816`'s directory entry. If there is no unused space, then the entry is appended to the end of the directory file. A quick analysis by the author suggests that an average of 56% of the final directory must be scanned before insertion is possible.[1] On average, 357KB of directory data is read before finding a suitable insertion point. (Both [Sweeney] and [Rakitzis] illustrate the wastefulness of linear scanning of large directories.) Furthermore, the insertion operation is synchronous: the change must be committed to disk before the open system call can return.

As a result of sequential directory scanning and synchronous file linking, the amount of time necessary for INN to write a single article file, regardless of the its size, can vary tremendously. Almost every piece of

[1] An unattributable Net rumor estimated 70%. The author's test sampled the `misc/jobs/offered` subdirectory every five minutes for a 24 hour period.

```
Path: news.mr.net!mr.net!visi.com!news-out.visi.com!ix.netcom.com!news
From: John Doe <jdoe@no.where>
Date: 2 Sep 1997 18:16:52 GMT
Newsgroups: misc.jobs.offered,mn.jobs
Message-ID: <foo87692.bar85-97@no.where>
Subject: French Fry Slicer (Level II) wanted for part-time work
Lines: 22
```

Figure 1: Sample headers from a sample Usenet article.

Level	Directory Name	Size	Number of Entries
1	/	512B	31
2	/var	512B	20
3	/var/spool	512B	12
4	/var/spool/news	15KB	994
5	/var/spool/news/misc	3KB	127
6	/var/spool/news/misc/jobs	174KB	10,836
7	/var/spool/news/misc/jobs/offered	637KB	32,499
5	/var/spool/news/alt	25KB	1367
6	/var/spool/news/alt/atari	512B	1
6	/var/spool/news/alt/fan	15KB	771
10	/var/spool/news/alt/swedish/chef/bork/bork/bork	512B	3

Figure 2: Sample directory sizes for an INN server spool filesystem

hardware in the server affects this measurement, though the number of disk drives spanned by the filesystem, the spanning type (e.g., none, RAID 0), the filesystem type, and the overall disk I/O workload of the drives are the largest factors influencing this measurement. It is not unusual to witness write times below 10 ms to over 200ms. INND, which is a single-threaded process, is blocked until open() is finished.

It should be noted that the sample article was cross-posted to another newsgroup, mn.jobs. INN will use a hard or symbolic link for `mn/jobs/8149` to link to the original article file, `misc/jobs/offered/9124816`. The linking process is identical to the file creation process with respect to directory file modification. Again, INND blocks during the link system call.

Another task INND performs is the processing of control messages. Article cancellation messages are by far the most frequently encountered control messages. To process a cancel message, INND performs a `history` database lookup to check if the message to be cancelled has arrived yet. If it has, the article is removed by INND. The amount of time this operation takes can vary from 8 ms to over 250 ms. Unlink() is also performed synchronously under FFS, which yet again halts INND's activity until the directory data is committed to disk.

Some Usenet servers are already seeing article volumes at or above 750K articles per day. A server must accept, on average, 8.68 articles/second, or 115 milliseconds/article, just to stay even with traffic flow. The 115 milliseconds includes network latency, server overhead (`history` database queries are the biggest INND factor here), and storage subsystem overhead. The measurements in Figure 8 and Figure 9 suggest that filesystem overhead and disk latency account for, at a bare minimum, an average of 50ms of overhead per article. The true value is probably closer to 75ms, and that's on a well-configured and otherwise idle server! Continued exponential article growth rates will only make this problem worse, particularly with a single-threaded server application.

An Article Is Forwarded

INND's `newsfeeds` file contains the rules for determining whether an article should be forwarded to one or more peer servers. Programs such as INNXMIT, INNFEED, and NNTPLINK are used to perform the actual article transmission.

INNXMIT reads a batch file containing the Message-IDs and filenames of articles to be forwarded to a particular Usenet peer server; there is at least one INNXMIT process per feed.[2] INNXMIT processes are run periodically by a shell script called "nntpsend", which is usually run via "cron".

Once INNXMIT has made an offer to send an article to a peer and the peer has accepted, INNXMIT's open() of the article triggers another flurry of OS activity. If lucky, the OS need not trigger any disk I/O because all of the required data is stored in-memory in the directory name and filesystem buffer caches, though the kernel CPU time spent searching the latter may be non-trivial. In the worst case, the example will require a linear scan of 830KB of directory data, triggering several disk I/O's to different filesystems. INNXMIT's batch file is written in the same order in which INND received the articles, adding a significant amount of randomness that renders the OS caches less effective.

The roles of NNTPLINK and INNFEED will be addressed in a later section.

An Article Is Read By Reader Clients

When INN was originally written, as with C News and its predecessors, Usenet client software accessed articles by reading the article files directly out of the spool directory. For a client such as "readnews" or "rn" to see what articles were available in "misc.jobs.offered", it simply had to generate a list of files in the `misc/jobs/offered` subdirectory. Usenet users had accounts on the Usenet server; client software ran on the server.

Today most Usenet reader clients use the NNTP protocol to access Usenet articles. INN includes a separate program, NNRPD, to handle communication with NNTP reader clients. Like many other UNIX-based servers, INND will fork() and exec() a child NNRPD process for each simultaneous reader session.

NNRPD has the same filesystem-related problems that INNXMIT does, though to a lesser degree. Filesystem buffer and directory name cache hit rates are better because NNRPD is much more likely to access a large number of article files in the same directory, avoiding cache churning. Also, FFS's tendency to group files in the same cylinder group also means that disk heads do not usually travel far to read files in the same subdirectory. Unfortunately, the heads are quickly relocated by other server activity, since most NNTP reader clients are run by humans who ordinarily take more than a few milliseconds to read a typical Usenet article. ''Sucking'' clients and automatic binary decoding utilities are growing in popularity, generating larger bursts of intensive disk operations than purely human-controlled software.

Another significant problem is that each article file access triggers an update of that inode's ''last access time'' value, which requires a disk write operation after each file is read. (The update also occurs when INNXMIT, NNTPLINK, or INNFEED reads a file to be forwarded to a peer.)

[2] An INN administrator may configure multiple feeds to a single peer, e.g., one feed of articles under 100KB in size and one for articles over 100KB.

An Article Is Removed

An article can be removed in several ways:

1. removed by a cancel control message.
2. removed by INN's "expire" process, usually via the "news.daily" script.
3. removed by a third party, e.g., an administrator using "rm -rf /var/spool/news/alt/binaries".

"Expire"'s list of files to remove is usually sorted, so close locality of reference helps achieve good cache hit rates. However, the final directory search is a sequential one (barring assistance from the directory name cache), requiring significant kernel CPU time to process. Each unlink() operation is synchronous, slowing the operation down much further: unlink() can take anywhere from 8ms to over 300ms to complete. If the article is cross-posted, all corresponding hard or symbolic links must also be removed (each at identical cost).

"Expire" is usually run early in the morning, when CPU utilization is lower, fewer articles are being transferred in/out (feeds across timezones can reduce this advantage), and fewer NNRPD processes upset the spool filesystem with additional disk activity. While "expire" is running, however, the disks(s) in the spool filesystem are effectively saturated with I/O requests: additional operation, such as sending/receiving articles, only slows down both the article expiration and the sending/receiving processes even further.

Improvements in INN Performance, a.k.a. Previous Work

A number of changes have been made to INN over the years to improve its performance. The following is a partial list of the more important ones:

1. Sort the list of article files to be removed and use "fastrm" to remove them.

   Sorting tremendously improves locality of reference in OS caches by removing all designated files in, for example, `misc/jobs/offered` at once rather than removing them in the order in which they arrived. Arrival order is typically quite random, which causes poor file buffer and directory name cache hit ratios when removing article files in arrival order. "Fastrm" first changes the working directory to the one containing the specified files; it calls chdir() using a relative path if shorter than a full path, and it gives simple filenames only to unlink(), which lowers namei() overhead.

2. Use the ‘‘-L’’ INND flag and "crosspost".

   The program "crosspost" performs the linking operations for cross-posted articles that INND would otherwise do. It doesn't accelerate the underlying filesystem operations, but it keeps INND from excessive blocking while waiting for the links to be made.

3. Mount multiple filesystems under spool directory.

   Originally done to provide the spool directory with more storage capacity than any single disk drive, it also provides a method of distributing disk I/O over multiple disk drives and disk controllers.

4. Make the spool directory a single virtual filesystem spanning multiple disk drives.

   Anyone who has tried to run an INN server receiving a full feed onto a single spool disk drive will discover that there aren't enough hours in a day to perform all the disk activity an FFS filesystem requires. Virtual filesystems can be implemented in software, e.g., Sun's Solstice DiskSuite or *BSD's and Linux's concatenated disk drivers (ccd and md, respectively), or in hardware, e.g., hardware RAID controller. This method has an additional benefit of creating a single pool of disk space, making it less likely that a burst of articles will fill the filesystem than improvement #3, as well as eliminating the need to use symbolic links across filesystems.

5. Sort INNXMIT batch files by filename instead of arrival order.

   Sorting INNXMIT batch files (in the "nntpsend" script) allows OS caches to be more effective for the same reasons sorting "expire"'s batches improves file removal rates. The author has experienced up to three-fold improvement in small article transfer rates by INNXMIT when using sorted batches, particularly if the batch is large and contains relatively few unique newsgroups. Experimental data can be found in [Delany].

6. Use NNTPLINK instead of INNXMIT.

   NNTPLINK runs concurrently with INND, one NNTPLINK process per peer feed. If the peer can accept articles quickly enough, NNTPLINK will read an article file within a second of being written. In this case, the odds are quite high that the OS still has all necessary information in its caches, avoiding the need for disk I/O. It also improves directory name cache performance, which in turn lowers directory scanning requirements.

7. Use INNFEED instead of INNXMIT.

   INNFEED operates similarly to NNTPLINK, but a single INNFEED process can feed multiple peers. Articles are read in their entirety into memory, then reference counts are used to avoid re-reading articles to be sent to multiple peers. A single process also helps reduce VM and context-switching workloads.

8. Avoid inode last access time updates.

   Linux users can use the ‘‘no_atime’’ mount option, and Network Appliance users can use

"option no_atime_update" [Swartz96] to avoid updating an inode's last access time attribute each time the file is read. Does a Usenet administrator really care about the last time an article file was read?

9. Use the "async" mount option.

   BSD users can mount their spool directory with the "async" option, which causes synchronous file operations to be performed asynchronously. Performance can improve dramatically but the risk of filesystem corruption in the event of a system crash is orders-of-magnitude higher. Linux's ext2fs uses asynchronous metadata updates by default; its careful ordering of disk writes greatly reduces the risk of filesystem damage. Legato's Prestoserve card negates this risk by making metadata updates into NVRAM, giving asynchronous-like behavior, then committing the changes to disk at a later time.

10. Use a dedicated, high-performance fileserver for spool storage.

    Use of high-performance fileservers such as Network Appliance can greatly reduce I/O delays [Swartz96] as well as serve the spool filesystem to multiple machines [Christenson]. Negative aspects include NFS and TCP/IP protocol overhead as well as potential network congestion and latency.

11. Use non-FFS-based filesystems.

    The XFS [Sweeney] and WAFL [Hitz] filesystems offer a number of enhancements that help minimize directory scanning requirements. Both, if the entire Network Appliance toaster is also considered, also minimize metadata update latency via write-ahead logging to disk or to NVRAM, respectively.

12. Multi-threaded NNRPD.

    Like INNFEED, a multi-threaded NNRPD reduces VM and context-switching requirements. An implementation in Java is already available [Poskanzer].

13. Avoiding use of INN altogether.

    Several Usenet software packages boast large performance increases compared to INN running on identical hardware. These include Diablo [Dillon], NNTPRelay [Sedore], KNews [Krten], and Cyclone [Highwind]. All of these packages avoid the standard "one article per file" storage method.

One innovation, the "streaming" extensions to the NNTP protocol, has not been formally defined, though it is widely used. It's primary benefit is reducing network-related latency in the non-streaming "IHAVE" lock-step dialog. If a server's disk subsystem is already at or near saturation, reducing network latency will only push drive utilization even higher.

INN Solution #1: File Naming by MD5 Hash

Perhaps first proposed in [Aguirre], this method was suggested to reduce file creation, access, and removal delays by making the INN spool directory "bushier," i.e., putting a limit on subdirectory depth and limiting the number of files in any particular directory. (With a standard INN server, it's common to run into directory depth extremes like those in Figure 2.) Instead of storing an article in a path based on a newsgroup + article number tuple, the Message-ID is hashed using the MD5 cryptographic hash algorithm [Rivest]. The result is converted to an ASCII hexadecimal representation and then modified using this algorithm. (See also Figure 3.)

1. Use 6 bits in the first byte as the first subdirectory name.
2. Use 6 bits in the second byte as the second subdirectory name.
3. Use the next 4 bytes for the filename.
4. In the event of a collision, append a "+" to the filename and test again for collision.

This method guarantees that any file's path would contain exactly two subdirectories and that each directory has exactly 64 subdirectories.

In April 1996, when this method was implemented, Usenet article volume was typically 180,000 articles/day. Subdirectories on a feeder machine storing articles for three days would contain approximately 125-135 files. Each filename was 8 bytes long, making each subdirectory consume about 2KB.

The MD5 hashing experiment taught the author a number of valuable lessons:

1. Article numbers are not required for feeder-only INN servers.
2. The standard `news/group/854` pathnames can be transformed into opaque "where is the article stored?" strings without affecting too many of INN's components.
3. Due to existing "history" and "expire" implementations and a desire not to meddle with those implementations, only a single expiration policy is feasible, instead of multiple per-hierarchy and per-group policies.

Message-ID	<foo87692.bar85-97@no.where>
MD5(Message-ID)	a0c277ebdad179a67c0203de8295269b
File path	20/02/77ebdad1

Figure 3: MD5 hash storage example

4. Disk I/O utilization, compared to a standard INN server on the same hardware, is cut roughly in half.

The MD5 hash version of INN was used in production on the Minnesota Regional Network's (MRNet's) three feeder-only machines for a period of about two months in mid-1996. Though the source code wasn't widely-distributed, it was used by at least one server in the Usenet Top 10 [Top1000] until approximately May of 1997.

INN solution #2: Article Storage in Cyclic Buffers

Filesystem research is an extremely active field of study. All UNIX filesystem improvements, from small changes in existing implementations to entirely new ones, must include functional characteristics such as the ability to create, modify, and remove files and their associated metadata, manage directory hierarchies, access control, concurrency restrictions, access auditing, and so on.

As pointed out in [Templeton96b], most of those characteristics do not apply to Usenet articles. Specifically, articles sizes are known in advance and do not grow during creation. Article text is never modified once committed to disk, simplifying space management policies. In addition, Usenet articles are stored for relatively brief periods of time, usually under two weeks, and large batches of articles, which arrived at similar times, are removed at the same time. If a Usenet administrator must store articles for longer, the number of articles is low enough that the large directories aren't a large problem or the articles are stored using an entirely different method, e.g., [DejaNews].

Cyclic Article Storage Method

The cyclic buffer article storage method was proposed in [Templeton96a]. When an article is accepted, the server decides which cyclic buffer will store the article. The server may choose between one or more buffers. The decision algorithm might consider the article's size, posted newsgroup(s), the "From" address, and/or other criteria. The article is written to the cyclic buffer at the location indicated by the "free" pointer. If the article doesn't fit within space pointed to by "free," the "free" pointer is reset to the beginning of the buffer before writing the article. After the article was written, the "free" pointer advances to a position beyond the end of the newly-written article.

In Figure 4, articles A-L were written inside the cyclic buffer. When the server decided to store article M within the buffer, it moved the "free" pointer back to the buffer's beginning before continuing. The "free" pointer points to the end of article P, the last written into the buffer. Article I is the oldest article still available in the buffer: articles M-O have overwritten A-G, and article P has partially overwritten article H.

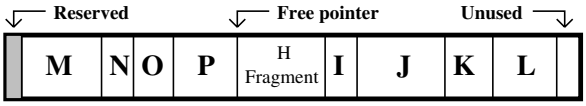

Figure 4: An example cyclic buffer

An INN Cyclic Buffer Implementation

The implementation described below is still considered experimental, though it has been in use at MRNet for over a year as well as on a couple dozen servers world-wide. The main design principles were simplicity and minimization of coding time. INN was chosen as a base to work upon, and few additional bells and whistles have been added. Support for NNTP reader clients was originally omitted from the design, though it has since been added.

The current source code is full of references to "raw disk partitions." When the author first began development, he naively assumed that "raw"/character disk devices would be the ideal storage medium, but their synchronous write semantics result in terrible performance. A "cooked"/block disk device (if the OS supports mmap()'ing block devices) or a large file, e.g., 2GB on top of a standard filesystem, works much better by allowing the OS to delay and consolidate writes. The word "raw," however, will remain until the code is rewritten.

Cyclic INN Article Processing

Once INND has accepted an article, it consults the article storage rules table. Its external, human-readable version is found in a new configuration file called "config.rawpart" (parsed at startup time); see Figure 5, section 3. The first rule matched is used. For the example article in Figure 1, the article will be stored in a "metarawpart" called SMALLARTS.

A "metarawpart" is a collection of one or more "rawpart," or cyclic buffer, components. If a metarawpart has more than one component, the server will choose one of them in round-robin fashion. See Figure 5, section 2 for configuration syntax.

Figure 5, section 1 lists three "rawpart" buffers. In this example, SD0 resides on an old Seagate Hawk 2GB drive. SD1 and SD2 reside on newer Seagate Barracuda 2GB drives. All three are partitioned identically, with one 2GB partition on slice zero. The assignment of large and heavily-cross-posted articles to a 2GB space and smaller articles into a 4GB space reflects the administrator's desired article retention policies. Articles written inside BIGARTS will recycle a couple of times per day, whereas articles written inside SMALLARTS will not be overwritten for many days.

Cyclic Buffer State Information

INND maintains an array of RAWPART structures that stores the state of each cyclic buffer (see Figure 6). While running, INND periodically converts the rawpart state data to ASCII and writes a RAWPARTEXTERN structure to the beginning of each buffer file to preserve buffer state information in the event of a process or system crash. No special efforts are made to synchronize the in-memory state data with the data on disk. It is therefore possible to lose a small number of articles in a crash: some may be overwritten when INND restarts. Similarly, only one copy of the data is written; there is nothing analogous to a ‘‘backup superblock.’’ This lack of redundancy has yet to cause a loss of state data in over a year of use.

The state data is written to disk in ASCII in a gesture toward platform-independence. This facilitates serving cyclic buffers via NFS to heterogeneous servers and copying buffers between servers.

Near the beginning of the buffer, between the RAWPARTEXTERN structure and the start of the actual article storage area, is a bitmap that records which 512-byte blocks within the buffer contain valid RAWARTHEADER structures. The bitmap is mmap()’ed to provide easy pointer access to its contents. The amount of space the bitmap uses, e.g., 512KB for a 2GB file, is small. In return, the bitmap makes it possible to determine if an article has been cancelled without attempting to read and validate its RAWARTHEADER structure.

The absence of the bitmap does not adversely affect a feeder-only INN server. Earlier implementations demonstrated that the additional I/O used to attempt to read cancelled article files is not significant. The article’s RAWARTHEADER structure is overwritten with the string ‘‘CANCELLED,’’ which foils further attempts to read the article. The bitmap’s role in the NNTP reader client support, as discussed later, solves an important performance problem.

INND gives an article, once it is accepted, a filename for use within other parts of INN. Unlike a standard filename such as `misc/jobs/offered/9124816`, the cyclic name looks like "SD1:24da800:1e". Colons separate the ‘‘rawpart’’ buffer name, offset within the buffer, and buffer’s current cycle number. Together with the information in `rawpart.config`, this name contains all of the information required to locate the article’s text. If the rawpart’s current ‘‘free’’ offset and cycle number are known, it is trivial to determine if the article has been overwritten by a later article. In this example, the RAWARTHEADER structure would be written at offset 0x24da800, followed immediately by the full text of the article.

A compile-time option determines if the article text is stored with the UNIX-style newline convention or stored in ‘‘wire format,’’ with CR-LF newlines and periods at the beginning of a line escaped with an additional ‘‘.’’ [Kantor]. Storage in wire format eliminates the need to convert newline styles each time the article is transmitted via NNTP, reducing user CPU time utilization in busy servers.

```
# Section 1
# Format: "metarawpart" (literally) : name : "I" (literally) :
#             comma-separated list of "rawpart" buffers
metarawpart:BIGARTS:I:SD0
metarawpart:SMALLARTS:I:SD1,SD2

# Section 2
# Format: "rawpart" (literally) : name : path : size (in hex!)
rawpart:SD0:/var/spool/dsks/c0t0d0s0:7A000000
rawpart:SD1:/var/spool/dsks/c0t0d0s1:7A000000
rawpart:SD2:/var/spool/dsks/c0t0d0s2:7A000000

# Section 3
# Format: "groups" (literally): rule or wildcard pattern: metarawpart name
# Store in BIGARTS if 1st group matches *warez* or *binaries*
groups:*warez*:BIGARTS
groups:*binaries*:BIGARTS
# Store in BIGARTS if article size is greater than 100KB
groups:~>100000:BIGARTS
# Store in BIGARTS if article is cross-posted to 10 or more groups
groups:~G10:BIGARTS
# Store all others in SMALLARTS
groups:*:SMALLARTS
```

Figure 5: The storage rules section of "config.rawpart"

NNTP Reader Support

The original cyclic buffer INN implementation did not support NNTP reader clients: the reader client reliance on article numbers greatly complicates an otherwise straightforward design. However, MRNet's main NNTP reader server, news.mr.net, started to show its article handling capacity limits in early 1997 – and the NNTP reader support was born.

INND is a busy process, so putting article number support back into INND was the last thing the author wished to do. Unfortunately, INND maintains low- and high-article values for each group inside the `active` file. Many NNTP reader clients rely on those values to be correct; incorrect values mislead many clients into believing that new articles have not arrived in a newsgroup when, in fact, new ones may be available.

A compromise was finally implemented. Most NNTP reader server administrators use "overchan", a program which creates Overview article summaries to eliminate the need for NNTP readers to retrieve the headers of all articles within a group (e.g., for discussion threading). "Overchan" was given the task of assigning article numbers. It also updates the mechanism that maps newsgroup + article number tuples => article storage locations and informs INND of newsgroup high-article value changes so INND can update the `active` file accordingly.

Several databases for the newsgroup + article number tuple => article storage location mechanism, such as GDBM and Berkeley DB, were considered. All were rejected in favor of a simpler mechanism: flat ASCII files with fixed-length lines.

Each newsgroup has an Overview file with a name like `misc/jobs/offered/.overview` (relative to the OVERVIEWDIR directory). A

```
/* Main internal data structure */
typedef struct {
  char          magic[RAWMASIZ];/* Magic string RAW_MAGIC */
  char          name[RAWNASIZ]; /* Symbolic name: 15 bytes */
  char          path[RAWPASIZ]; /* Path to file: 63 bytes */
  RAWPART_OFF_T len;            /* Length of writable area, in bytes */
  RAWPART_OFF_T free;           /* Free offset (relative to byte 0!) */
  struct metarawpart *mymeta;   /* Pointer to my "parent" metarawpart */
  time_t        updated;        /* Time of last update to header */
  int           fdrdwr;         /* O_RDWR file descriptor, "innd" use only! */
  int           fdrd;           /* O_RDONLY file descriptor for this rawpart */
  CYCLENUM_T    cyclenum;       /* Number of current cycle, 0 = invalid */
  int           magicver;       /* Magic version number */
  int           articlepending; /* Debug flag: article is pending for write */
  caddr_t       bitmap;         /* Bitmap for article in use */
  RAWPART_OFF_T minartoffset;   /* The minimum offset for article storage */
} RAWPART;
extern RAWPART rawparttab[MAX_RAWPARTS];

/* Main data structures written to disk */
typedef struct {
    char        magic[RAWMASIZ];
    char        name[RAWNASIZ];
    char        path[RAWPASIZ];
    char        lena[RAWLASIZ];         /* ASCII version of len */
    char        freea[RAWLASIZ];        /* ASCII version of free */
    char        updateda[RAWLASIZ];     /* ASCII version of updated */
    char        cyclenuma[RAWLASIZ];    /* ASCII version of cyclenum */
} RAWPARTEXTERN;

typedef struct {
  long          zottf;          /* This should always be 0x01234 */
  long          size;           /* Article size, converted by htonl() */
  char          m_id[64];       /* 63 bytes of Message-ID should be enough */
} RAWARTHEADER;
```

Figure 6: Important on-disk and in-memory data structures

newsgroup + article number => storage location mapping file called `name.map` is created in that same directory. The example in Figure 7 shows that the first map entry begins at byte 102, the low article number is 817053, the file was last updated on 2 September 1997, and that the group has at most three articles (which may have been cancelled or overwritten).

```
BodyStart   102
LowArt      817053
LastRewrite 873194497
SD1:10a2a000:2a
SD2:1b9b2800:18
SD1:11527400:2a
```

Figure 7: A sample "name.map" file for `misc.jobs.offered`

The `name.map` file format makes additions or changes easy. Each line in the `name.map` file is exactly 34 bytes long (including newline), making offset calculations for an individual map entry simple. NNRPD, when it receives a ‘‘GROUP’’ command, mmap()’s the newsgroup’s `name.map` file for pointer-based access to the file. The file is then read, and the appropriate bitmaps are checked to determine which articles still exist; this replaces the readdir() loop a standard NNRPD process uses to create the "ARTnumbers" array of existing articles.

Ordinarily it is sufficient to know only the low- and high-article values for a newsgroup. However, it is necessary in one situation to know exactly which articles have been cancelled: when sending Overview information to the client. If all Overview data between article numbers ‘‘low’’ and ‘‘high’’ are sent to the client, the client believes that all articles in between are accessible. The user is shown on-screen that a cancelled article is accessible, but an attempt to retrieve the article text will fail. MRNet’s users considered this to be a bug, not a feature.

Early implementations required reading each RAWARTHEADER structure to determine if an article had been cancelled, which generated an enormous disk I/O burden and caused huge delays in NNRPD’s Overview responses. The bitmap was added to each ‘‘rawpart’’ buffer to solve this problem.

Like the Overview’s `.overview` files, the `name.map` files require periodic pruning to remove references to articles that have been expired (standard INN), cancelled, or overwritten (cyclic INN). A Perl script called "expire.mapprune" performs this task; like "expireover", it is typically run once per day. To update the file without rewriting it, the ‘‘BodyStart’’ line is simply overwritten with a larger value.

Performance Observations and Evaluation

Until now, most of the evidence of a cyclic INN server’s superior performance has been anecdotal. Examples include:

- Use of "sar" and "iostat" to watch disk activity. MRNet’s central feeder machine, when using a standard INN server, was engaged in about a dozen bi-directional feeds that kept its disk drives anywhere from 20-50% busy on a sustained basis; during article expiration periods, drive activity was sustained at 70-90% busy. A cyclic INN server and 70 one-way feeds can rarely sustain a 25% busy workload.
- A Pentium 120MHz PC with 128MB RAM, one SCSI controller, three 5400 RPM disks (two for article storage), and a 100Mb/s Ethernet interface is configured with 45 one-way feeds to MRNet members. Several times a member’s Usenet server has been down for several hours, and when the server is back online, MRNet receives telephone calls from the members trying to trace a denial-of-service attack. The FreeBSD PC, resuming the Usenet feed, has been the source of the ‘‘attack.’’
- A Sun SPARCstation 5 with an 85MHz CPU, 192MB RAM, one SCSI controller, three 5400 RPM disks (two for article storage), and a standard 10Mb/s Ethernet interface was able to climb as high as 20th place in the Usenet Top 1000 rankings [Top1000].
- The performance improvement in cyclic INN has allowed MRNet to avoid buying faster equipment for Usenet servers for over a year while providing superior Usenet feeds to its members and customers ... with the periodic exceptions of using code that wasn’t quite as stable as it should have been.

Most of the observations and measurements in this section are limited to INND’s operation because its function is the easiest to measure and to control tightly. The measurements aren’t intended to be a rigorous investigation into a cyclic INN’s performance; rather, they are a first attempt to quantify how much this implementation lowers disk I/O activity.

Experimental Platforms and Methodology

In an attempt to systematically measure system performance while INND accepted articles, four separate batches of articles were sent to two INN servers. The servers were idle, with the exception of INND’s activity: all INNXMIT, INNFEED, NNRPD, and other non-INN-related processes were killed. The `history` file was truncated to zero bytes, followed by a "makehistory -i -r -s 500000"; a large `history` DBZ index file can introduce a significant amount of latency which would skew the test results. The reference INND process was run with the ‘‘-L’’ flag, and "crosspost" was not used to perform the additional linking work. Each server was monitored for CPU utilization, file buffer and inode cache hit rates, and disk utilization using a custom script based on an example from the SE Performance Toolkit [Cockcroft]. The results of all four batches were averaged and shown in Figure 8 and Figure 9.

The reference INN server is a Sun SPARCstation 10 with two 100MHz hyperSPARC CPUs, Solaris 2.5.1, 272MB RAM, a 100Mb/s Ethernet interface, and eight 7200 RPM disks spread across three SCSI controllers. The five disk drives storing the spool filesystem are 4GB Seagate 15150W (“Barracuda 4” Fast & Wide SCSI) all connected to the third SCSI controller, a Sun Fast/Wide SCSI SBus controller. The cyclic INN server is a Sun SPARCstation 5 clone with a single 85MHz microSPARC II CPU, Solaris 2.5, 128MB RAM, built-in 10Mb/s Ethernet interface, and two 5400 RPM disks on the built-in SCSI controller. A single 2GB Seagate ST32430N (“Hawk 2LP” Narrow SCSI) stored the server’s single cyclic buffer, which is accessed via a block disk device (rather than a file on top of a UFS filesystem).

The server sending the articles is an Intel PC with a single 200MHz Pentium Pro CPU, FreeBSD 2.1.7.1-RELEASE, 256MB RAM, a 100Mb/s Ethernet interface, and three 4GB 7200 RPM disks on two SCSI controllers. Two of the drives, Seagate 15150Ns, store three cyclic buffers consisting of 2GB files stored on top of two FFS filesystems. The batches of articles it sent were from 5,300 to 5,800 articles each, posted to approximately 3,000 different newsgroups. Each article was less than 32KB in size in order to emphasize article throughput; the average article size was about 2.6KB. The sending machine was under a relatively heavy load: INN was accepting articles from 95 to 100 NNTP sessions, and three INNFEED processes were feeding articles back to 38 peer servers. INNXMIT was used to transmit each batch in a single streaming NNTP session.

The article throughput results in Figure 8 are dramatic. The cyclic INN server accepted articles three to four times faster than a standard INN server, despite the fact that the standard INN server had more CPU, RAM, disk, and network resources available.

The data in Figure 9 are even more dramatic. Spool disk activity across the board is reduced by approximately an order of magnitude. When the facts that there is no explicit time- and resource-consuming article expiration process and that disk I/O for article

Batch number	**1**	**2**	**3**	**4**	**Total**
Articles in batch	5,301	5,318	5,826	5,407	21,852
Unique newsgroups in batch	2,961	2,962	2,895	2,616	-
Size of batch (KBytes)	13,537	12,193	15,485	15,571	56,786
Elapsed time by standard INN (seconds)	480	486	551	530	2,047
Articles/second accepted by standard INN	10.1	10.1	9.9	9.4	-
Elapsed time by cyclic INN (seconds)	153	168	135	141	597
Articles/second accepted by standard INN	31.7	29.3	40.3	35.2	-
Article throughput increase factor	3.13	2.89	4.08	3.76	-

Figure 8: Batch article transmission results.

Statistic	**Standard INN**	**Cyclic INN**	**Δ Relative to Standard**
Average CPU idle time	26.41	38.43	46 %
Average CPU user time	13.39	40.09	199 %
Average CPU system time	12.63	12.71	1 %
Average CPU wait time	47.57	8.77	-82 %
Read() + readv() calls	192,280	28,622	-85 %
Write() + writev() calls	367,806	136,368	-63 %
Pathname lookups	35,325	3,606	-90 %
Ufs_iget() calls	26,592	65	-99 %
Spool disk data read (KB)	153,575	50,116	-67 %
Spool disk data written (KB)	519,802	75,688	-85 %
Spool disk reads issued	25,258	1,245	-95 %
Spool disk writes issued	88,600	8,059	-91 %
Spool disk average service time (ms)	48	13	-73 %
Buffer cache hit rate	92.63	98.37	6 %
DNLC hit rate	65.74	93.01	41 %
Inode cache hit rate	11.77	16.20	38 %

Figure 9: Kernel statistics taken during article transmission.

cross-post links was not measured are taken into account, the decrease is more than a factor of 10.

The file buffer and directory name cache hit rates for the standard INN server are quite good, which is consistent with it having a large amount of RAM available and with the system performing no other substantive tasks while the measurements were taken. The amount of data the cyclic INN server read from the spool disk is conspicuously high; though article cancellations and reads of the mmap()'ed buffer bitmap play a role, further study is required to find a satisfactory explanation.

In a less-tightly controlled test, a transfer of a batch of 20,000 articles, all 4KB in size or less, between two Sun SPARCstation 5s which both use cyclic INN, exhibits sustained transfers of 100-150 articles/second (when the `history` file is first truncated) and occasionally peaks to 200 articles/second. Though this test was not performed on the reference INN server, the author would not anticipate that the server would exceed 25 articles/second.

The code supporting NNTP reader clients is still under development. The addition of an article to a `name.map` file involves opening the file, reading its first few lines, then appending a single line to the end of the file. The disk I/O for these operations should compare quite favorably to an expensive file linking process. Updates for multiple articles which arrive shortly after the first in the same newsgroup may be consolidated into a single operation. Reading an article by number involves opening the `name.map` file, reading the first few lines, seeking directly to the corresponding entry, reading a line's worth of data, seeking to an offset within a cyclic buffer, and one (or more) data reads. Access to subsequent articles in the same group does not require re-reading the head of the map file. Article expiration on a cyclic INN server involves rewriting the `name.map` file, removing unwanted lines in the process.

Future Work

Because the cyclic buffer code is still experimental, a number of changes and additions remain to be done. A partial list, in no particular order, includes the following:

1. Incorporate the cyclic buffer code into the official INN source distribution, rewrite outstandingly-ugly hacks and ''temporary'' code as needed, and debug all components thoroughly.
2. Write a cyclic buffer library conforming to the INN soon-to-be-released ''storage manager API.'' This programming interface will make multiple article storage methods (e.g., traditional spool, cyclic buffer, commercial database) much easier to write and maintain.
3. Pursue further improvements in the ''history'' database mechanism. The current DBZ implementation does an excellent job of putting a low upper-bound on the disk I/O required to query the database, but its average amount of disk I/O per query is too high. Servers that receive more than a few million article offers per day perform too much disk I/O in the current implementation.
4. Pursue further research into the NNTP reader newsgroup + article number => storage location mapping mechanism. In particular, replacing "overchan"'s current communication channel with INND would be a major improvement.
5. Add an option to make append-only buffers and allow INND to create such buffers as necessary. Some administrators wish to store articles for a guaranteed period of time. When articles stored within an append-only buffer are due to expire, the buffer file is removed in a single unlink() operation [Krten].

Concluding Observations

Before writing this paper, the author had never systematically measured the operating differences between a standard INN server and a cyclic one. The results surprised even him. Though the measurements and analysis presented is not comprehensive, it makes a compelling case for changing INN's article storage model. The implementation should be optimized further, and the NNTP reader client support requires debugging and much more real-world stress testing before this implementation can be considered for general use.

As the Internet continues to grow in popularity, the Usenet will certainly grow along with it, and that growth is not likely to drop far below current rates. Any reduction in ''excessively-posted messages'' will be offset by additional people using Usenet. The underlying store-and-forward mechanism does not scale well, ignoring problems with specific implementations: the current model of ''broadcasting'' articles to all Usenet servers is inefficient. Caching NNTP servers, such as [Assange], and automated article cancellation programs, such as [CM], are partial solutions at best. However, techniques such as IP multicast [Lidl], or World Wide Web-like centralized article storage are years away from acceptance. The Usenet will continue to require software optimizations and faster hardware until a better distribution mechanism is widely-implemented.

Availability

Cyclic buffer INN was first used at MRNet in July 1996 and development has proceeded, with fits and starts, since then. By the time this paper is published, the cyclic buffer code may already be merged into the official INN source distribution, albeit in a development branch of the source tree. Until then, the code is available via anonymous FTP under INN's original licensing restrictions (which are quite liberal) at ftp://ftp.mr.net/pub/fritchie/cnfs/. Also available in

that directory is the SE script used to monitor the servers' utilization stats and the raw data it generated. A mailing list for CNFS discussion is available by sending "subscribe cnfs" to <majordomo@ mr.net>.

The official INN source distribution is available at ftp://ftp.isc.org/isc/inn/.

Acknowledgments

This paper and the work it describes would not have been possible without the support and encouragement of MRNet Engineering Department staff, not to mention the typing, which saved a lot of wear and tear on sore hands. This paper would still be only partially-fired synapses if Lee Damon hadn't mentioned that the LISA '97 abstract submission deadline had been extended. Many people have assisted with the cyclic ideas and prototype implementations, from stress-testing to bug-fixing: Jerry Aguirre, Michael Beckmann, Barry Bouwsma, James Brister, Matt Bush, Evan Champion, Mark Delany, Avi Freedman, Darrell Fuhriman, Jeff Garzik, Joe Greco, Dave Hayes, Chris Halverson, John Ladwig, Clayton O'Neill, Alexis Rosen, Rich Salz, Robert "RS" Seastrom, Sang-yong Suh, Brad Templeton, Jeff Weisberg, Sven-Ove Westberg, and many others unintentionally omitted. Mike Horwath provided the reference INN server, since MRNet no longer has a standard INN machine. Special thanks to Dave Diehl, Olaf Hall-Holt, Andy Mickel, Joe St. Sauver, Peter Seebach, and John Sellens for their manuscript reviews. A case of Leinenkugel's goes to Nick Christenson for his extraordinary critiques. Finally, to the Norge '96 crowd, Louise Lystig Fritchie in particular, thank you for your support in this continuing madness called Life.

Author Information

Scott Lystig Fritchie graduated with a degree in mathematics and a concentration in computer sciecne from St. Olaf College, home of Minnesota's first UNIX machine and of "stolaf," the state's one-time gateway to the rest of the Usenet. Scott's other claim to (obscure) fame is implementing one of the world's first World Wide Web interfaces for a library cataloging system; it has since mutated into WebPALS, used by the PALS Across Georgia project and the Minnesota State Colleges and Universities (MnSCU). He currently works as senior systems administrator at the Minnesota Regional Network (MRNet) and can be contacted via email at <fritchie@mr.net>.

Bibliography

[Aguirre] J. Aguirre. Posting to news.software.nntp: "File by message ID instead of group/number." Message-ID <4gl99l$blm@olivea.ATC.Olivetti.Com>, Feb 23, 1996.

[Assange] J. Assange and L. Bowker. NNTPcache. ftp://suburbia.net/pub/nntpcache/.

[Christenson] N. Christenson, D. Beckemeyer, and T. Baker. "A Scalable News Architecture on a Single Spool." In *;login:,* Vol. 22, No. 3, June, 1997.

[CM] Cancelmoose. NoCeM. http://www.cm.org/.

[Cockcroft] A. Cockcroft. The SE Performance ToolKit. http://www.sun.com/960301/columns/adrian/ .

[Collyer] G. Collyer and H. Spencer. "News Need Not Be Slow." In *Proceedings of the Winter 1987 USENIX Technical Conference,* Washington, DC, January, 1987.

[DejaNews] DejaNews. http://www.dejanews.com/.

[Delany] M. Delany. Posting to news.software.nntp: "cyclic news file system – more performance results." Message-ID <4lqh5o$kd6@bushwire.mira.net.au>, April 26, 1996.

[Dillon] Diablo: a backbone news transit system. http://www.backplane.com/diablo/.

[Graham] J. Graham. *Solaris 2.X: Internals & Architecture.* McGraw-Hill, 1995.

[Highwind] Cyclone NewsRouter. http://www.highwind.com/.

[Hitz] D. Hitz, J. Lau, and M. Malcolm. "File System Design For an NFS File Server Appliance." In *Proceedings of the USENIX Winter 1994 Technical Conference,* San Francisco, CA, 1994.

[Kantor] B. Kantor and P. Lapsley. "Network News Transfer Protocol." RFC 977, U. C. San Diego and U. C. Berkeley, February, 1986.

[Krten] R. Krten. "Improving Usenet News Performance." *Dr. Dobb's Journal,* May, 1996. See also: http://www.parse.com/ .

[Leffler] S. Leffler, M. McKusick, M. Karels, and J. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, 1989.

[Lidl] K. Lidl, J. Osborne, and J. Malcolm. "Drinking from the Firehose: Multicast USENET News." In *Proceedings of the USENIX Winter 1994 Technical Conference,* San Francisco, CA, 1994.

[McKusick] M. McKusick, W. Joy, S. Leffler, and R. Fabry. "A Fast File System for UNIX." *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August, 1984.

[Poskanzer] J. Poskanzer. Multi-threading nnrpd and caching proxy. http://www.acme.com/java/software/Package-Acme.Nnrpd.html.

[Rakitzis] B. Rakitzis and A. Watson. "Accelerated Performance for Large Directories." *Technical Report 3006*, Network Appliance, Inc.

[Rivest] R. Rivest, "MD5 Digest Algorithm." RFC 1321, MIT and RSA Data Security, Inc., April 1992.

[StSauver] J. St. Sauver. "The 1996/97 Oregon Christmas USENIX Newsadmin Newsletter." http://darkwing.uoregon.edu/~joe/pdf/.

[Salz] R. Salz. ''InterNetNews: Usenet Transport for Internet Sites.'' In *Proceedings of the USENIX Summer 1992 Technical Conference*, San Antonio, TX, June, 1992.

[Sedore] C. Sedore and E. Sedore. NNTPRelay Usenet News propagator. ftp://ftp.maxwell.syr.edu/nntprelay/ .

[Swartz93] K. Swartz. ''Forecasting Disk Resource Requirements for a Usenet Server.'' *Proceedings to the Seventh USENIX Systems Administration (LISA VII) Conference,* Monterey, CA, November, 1993.

[Swartz96] K. Swartz. ''The Brave Little Toaster Meets Usenet.'' *Proceedings to the Tenth USENIX Systems Administration (LISA X) Conference*, Chicago, IL, September, 1996.

[Sweeney] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. ''Scalability in the XFS File System.'' In *Proceedings of the USENIX 1996 Annual Technical Conference*, San Diego, CA, January, 1996.

[Tanenbaum] A. Tanenbaum. *Operating Systems: Design and Implementation.* Prentice-Hall, 1987.

[Templeton96a] B. Templeton. Posting to news.software.nntp: ''How to do a better USENET file system.'' Message-ID <DnCo3D.Cz3@clarinet.com>, Feb 25, 1996.

[Templeton96b] B. Templeton. Posting to news.software.nntp: ''Re: How to do a better USENET file system.'' Message-ID <DnFDJM.CMw@clarinet.com>, Feb 27, 1996.

[Top1000] Top 1000 Usenet sites survey. http://www.freenix.fr/top1000/.