Combined Log System

David Beckett[1], Computing Laboratory, University of Kent, Canterbury, CT2 7NF, England
D.J.Beckett@ukc.ac.uk, http://www.hensa.ac.uk/parallel/www/djb1.html

Abstract:

Busy Internet archives generate large logs for each access method being used. These *raw* log files can be difficult to process and to search. This paper describes a system for reading these growing logs, a *combined log file* format into which they are re-written and a system that automates this building and integration for multiple access methods. Automated summarizing of the information is also provided giving statistics on accesses by user, site, path-name and date/time amongst others.

Keywords:

archives, administration, statistics

Introduction

In a large Internet archive site, providing multiple methods of access (*ftp*, *gopher*, *WWW*, ...), there are a lot of raw log files being continually generated by the processes that handle the methods. Several programs exist to scan and summarize these different raw log formats for individual[myers94][fielding94] and multiple[hughes94][magid94] methods, but none does this in an extendible way.

For archive administrators, a better way is required to handle these raw logs and a log processing system is required that has these good design features:

Standard log format
: Uses a combined log file format, which has all relevant data retained from the raw logs, giving quick access to (at least) data by file name, user name, site name and type of access. Each entry should consist of one line formatted to make it easy to process with standard UNIX tools.

Logs stored chronologically
: Access to the logged transfers available; indexed by date and time.

Log summaries
: Summaries provided of (possibly older and compressed) information so that it doesn't need to be re-scanned for totalling byte counts etc.

Active raw logs
: Can handle growing raw log files being written to concurrently with the scanning.

Log rotation
: Can cope with raw log files being renamed, moved or rotated between scans.

Compressed files
: Is able to read and write gzipped and compressed old raw log files and previously processed logs.

Extendible
: Is very simple to add new raw log file formats.

Efficient
: Does not require excessive amounts of processing, storage or time when working (hopefully).

Design

The combined log file format was very important, based on the goals outlined above and thus was the first thing to be designed in detail.

Each entry corresponds to a single transfer of data (*access*) and needs fields to store all relevant pieces of information for each access type. These fields compose a single line of a combined log file.

Combined Log File Fields

The following fields were identified:

Type
: The access type of the raw log file being summarized (such as ftp, gopher, etc.) This needs to be encoded in every field so that information can be catgorized by type. Valid types are configurable. Mandatory field.

Operation
: The operation being performed. Most operations result in the transmitting of a file although other *pseudo-operations*, which don't involve a transfer, such as the start and end of interactive sessions can also be performed. Valid operations depend on the **Type** field. Mandatory field.

Date and Time (Datetime)
: The date and time of the access. Since the entries are going to be sorted by this field, it is important that it is easy to sort. Thus the following format was used: *YYYY-MM-DD-hh*:*mm*:*ss* where *YYYY*, *MM* and *DD* is the date (year, month, day) and *hh*, *mm* and *ss* is the time (hour, minute, second). This representation of the date and time makes sorting very simple - just using string comparisons which makes it easy for other programs/languages to process the output.

  The full date is needed, including the year and century. Sometimes this needs to be interpreted if only the last two digits of the year are encoded in the raw logs.

  The date component of this field is required however the time may not be known and if this is the case, it should be set to the (illegal) value "99:99:99".

Name (or Path)
: The name of the entry being transfered (if applicable). This may be a name referring to a file and if it is, it should be a full path name if possible. If the name is not a file reference it is a string that can identify the transfer, for example a URL. Optional field (but mandatory for transfer operations).

Size
: The amount of data, in bytes, transferred as a result of this access. If this is duplicated in another field, this can be represented by the number being bracketed, for example "(100)". This is an optional field since some some logs don't give the byte count transferred although this may be interpreted later.

User
: The user identified with the transfer. Optional field.

Site

The site name (or IP address) identified with the transfer. Optional field (but mandatory for transfer operations).

Email

The email address of the user identified with the transfer. The user and site fields may be empty if this field encodes both values as *user@site* or may be *user@* to imply *user@site*. Optional field.

Some of the above fields are optional, but require a place-holder to represent their absence. The place-holder was defined to be "-", that is, the minus sign character (ASCII 45).

These fields were given a physical encoding, as a single text line, composed from the concatenation of all the fields above, in the order given, with a single TAB (ASCII 9) character as separator and terminated with a line feed (ASCII 10).

There are a few restrictions to the field contents: no field may contain the TAB (ASCII 9) or space (ASCII 32) character except for the email field since is the last one on the line. In the future, these restrictions may be lifted by using an encoding, for example, the URL one "%" plus two hex-digits for 7-bit ASCII.

Example from the log for January 1994 for the Parallel Computing archive[2] (anonymized for site and user):

```
ftp     txfile  1994-01-19-11:27:14     /ftp/pub/parallel/documents/in\
mos/archive-server/checkocc/test80xa.occ  58019   -       123.45.67.89\
    abcdef@ghijklmn.fr
gopher  txfile  1994-01-19-11:27:39     /ftp/pub/parallel/parlib/butte\
rfly/queens/bflyparqueens.c       4789    -       abc.def.Uni-ghijk.DE\
    -
http    txfile  1994-01-19-11:27:54     /usr/l/lib/httpd/htdocs/parall\
el/home.html      961     -       unix.hensa.ac.uk        -
```

where the white spaces are TABs and `\` are line wraps. In this case, the lines represent transmitting a file - the `txfile` operation - for each method.

Combined Log Files

The lines representing the entries converted from the raw log files are then stored in files. These should then be indexed by date and time. This date-sorted information could be stored in a special database but for ease of use with standard (UNIX) tools, it was decided that the lines would be written into plain text files, with a range of dates applying to a file. The range of dates stored in any one file has several options:

Option	Output log file name
yearly	*YYYY*
monthly	*YYYY-MM*
daily	*YYYY-MM-DD*
monthly/	*YYYY/MM*
daily/	*YYYY/MM/DD*

These give the choice of either a flat or deep hierarchy of log files, stored by year, month and/or day as required. If the name format contains a "/" then sub-directories are used as appropriate. The choice may also be made depending on the size of the output files generated.

Inside each file, the information needs to be sorted by date and time but this needs only to be done occasionally, at worst once a day since that is the smallest date quanta in a single log file.

System Design

In the UNIX tradition, the system was designed as a circuit of communicating programs (some filters), passing data via pipes or files as the user prefers. The input to the system is raw log files, it works with combined log files and outputs these and summary files. The overall picture is shown in Figure 1: PostScript [B&W] or GIF [578x777, 1 bit]

The programs in the system are:

lscan
: Reading raw log files and writing combined log files.

lsort, lclean and lsqueeze
: Sorting, cleaning and gzipping / compressing combined log files in-place respectively.

sum-counts
: Summarizing combined log files by numeric fields and writing a summary file.

sum-names
: Summarizing combined log files for text fields and writing a summary file.

sum-sort
: Sorting summary files in place.

sum-format
: Reading summary files and outputting text/HTML[conolly95] documents.

Creating Combined Log Files (lscan)

The major problem in creating these combined log files from the raw logs is caused by the raw logs continually growing as the software daemons append to them. The new entries must be added as they appear at the end of the raw log files, beginning from where the last scan finished. It was also necessary to handle the log files being rotated (renamed), moved into other directories, and being compressed (gzipped) which are commonly done on these large files to save space. This required some careful thought and state saving between parses of the logs.

The system is configured to know, for each type of access:

- The latest log file being written to;
- The type of the log (wuarchive ftp, CERN http, NCSA http, etc.);
- How the logs are rotated, truncated or renamed;
- How to find the rotated log files - these may be compressed;
- ... and other flags.

The combined log files that have just had the newly added entries appended, are then be sorted by date and time in place, to preserve their internal order. These may then be compressed and then possibly summarized by one or more fields to present the information to the user.

The lscan program performs the creation process and for *each* type of raw log file, it does the following:

1. Find out where the parse finished for the previous scan, by checking a status file. If the log file has been rotated, a search must begin to find where the file now is. The last position may be found by checking in the older rotated logs or by searching line-by-line.
2. Convert each access into the combined log entry format. It is crucial to generate a date/time entry for each entry since that is the major sort field. This may involve some heuristics if, for example, the full year is not encoded in the raw log (e.g. gopher).
3. Clean up the resulting entry - ignore excluded path names, errors etc.
4. Append the entry to the correct file in the combined log file tree.

After all the combined log files have been updated, they should then be processed in place by lsort which sorts the entries in the files by date and time. They can also be compressed in place using lsqueeze to save disk space.

Summarizing combined log files

Once the information has been put in the combined log file format, it can then be summarized. This is equivalent to indexing by some fields in the log file, in database terms, but for this specialized case it was decided that simpler programs could be written and used rather than needing a full database.

The summarizing in this case consists of summing the byte and access counts indexed by

- Date and Time (sum-counts)
- A textual field (sum-names) eg user, site and path names.

The output of the summary, a *summary file*, can then be formatted and presented to the user as ASCII text or HTML output.

Summary Files

Since each summary file has potentially a different number of fields, this must be encoded in the summary file. Other information to encode is: the period (Datetime) covered by the summary; the totals for the byte and access counts; the number of data entries and an indication of the sort field if the data has been sorted.

This gave the following design for the elements in a summary file:

`period` *start datetime end datetime*
: The datetime (format as described earlier) period over which this data has been collected. This **must** be the first element - it is currently used by all summary programs to recognise a summary file from a combined log file.

`fields` *fields*
: The field names separated by a space. Mandatory element.

`field-widths` *widths*

The width of each field, separated by a space. This can be calculated during processing and remove duplication of work for later programs. Optional element.

`sort-field` *sort field name*

The name of the field by which this data was sorted. This is not used for the sum-counts program output. The type of the sort field determines whether the sorting will be done numerically or alphabetically. Optional element - when missing implies unsorted data.

`totals` *total access counts total bytes*

The totals of the numeric data which could be used later for further processing. Optional element.

`entries` *number of entries*

The number of data entries following. Optional element.

`data` **..**

The data summarized - space separated data corresponding to the fields described in the fields element above. These **must** be the last entries in the summary file, and none of the above elements must appear after the first data element. Mandatory element (if there is any data).

Summary File Operations

`sum-names` program

Summarizies the byte **OR** access counts with respect to any text field such as the name (path), email or site fields. In addition, the program can alter the site to be either an *institution* - a guess of the `real' site or a *country* and can reverse the site to give a reversed-domain name.

`sum-counts` program

Summarizes the byte counts and access count fields. It outputs a file indexed by date *scheme* which are:

scheme	scheme name	scheme values
per hour of the day	per_hour	00 to 23 (or ?? if not known)
per day of the month	per_day	01 to 31
per month of the year	per_month	01 to 12
date	date	*YYYY-MM-DD*
month-year	month	*YYYY-MM*
year	year	*YYYY*
total	total	-

The fields output are the *scheme name* followed by the *scheme value* and then the byte/access counts for each type seen.

`print-entries` program

Both of the above programs work on complete log files (or work as filters) but often a summary is required over a particular date period that doesn't correspond to whole combined log files. In this case, this program can be used to output the entries for a given period and this output, which is a combined log file, can then be piped into one of the above summary programs (or stored in a temporary file).

`sum-sort` program

Sort a summary file by any field - this only makes sense for data produced by sum-names since sum-counts outputs data already sorted by scheme and scheme-value.

`sum-format` program

Print the data prettily, either as text or HTML. It also allows a ranking to be given, for `top 10s' and percentage of the totals to be calculated for each entry.

`print-scheme` program

Print a particular scheme, from a summary-by-count for example, this is the *total* scheme for January 1994:

```
Data Period: 1994-01-01-00:56:55 to 1994-01-31-23:16:29
Data Summary for scheme: total
Type        ||            bytes   %bytes  |  Accesses   %Acc.  | Avg. Xfer
------------------------------------------------------------------------
ftp         ||      296,970,244    88.37  |     5,494   60.92  |   54,054
gopher      ||       38,103,232    11.34  |     3,380   37.48  |   11,273
fbr-howftp  ||       (2,772,384)   (0.82) |        (11)  (0.12) | (252,035)
fbr-email   ||         (934,670)   (0.28) |         (9)  (0.10) | (103,852)
http        ||          661,115     0.20  |       132    1.46  |    5,008
mserv       ||          319,060     0.09  |        12    0.13  |   26,588
fbr         ||            7,188     0.00  |         1    0.01  |    7,188
------------------------------------------------------------------------
total       ||      336,060,839   100.00  |     9,019  100.00  |   37,261
```

From this it is easy to see the most common access method at that time was ftp with World Wide Web http entries (new at the time) just starting up. The final column is the average transfer size which, as could be expected, gives much smaller values for http than the other methods.

`build-sums` program

This builds a cache of summaries for the current log files and generates super-summaries by month, year and in totals. This means a complete running total of all the statistics required over the entire life of the archive can be kept. It supports keeping up-to-date summaries for many types - count, site, country etc.

Other programs

Several auxiliary programs were also written to work on combined log and summary files including: sum-grep to do a pattern match in the output of summary-by-name files - it has to be used to preserve the totals; and lgrep for a similar operation on combined log files.

Results

At HENSA Unix[3], the system has been keeping up-to-date summaries of all the transfers since the archive was opend - currently (February 1995) over four years of logs, 300 gigabytes of data sent and 10 million accesses are kept up to date.

With a concrete design like this, there are likely to be missing things that need to be added later. An example of this is the result code returned by the HTTP daemons (amongst others). Since no result field existed, it was appended to the operation field where it can be found if needed. Since most operations succeed, it makes the failed ones stand out:

```
http    txfile/fail=404 1995-02-01-02:25:06     /ftp/pub/parallel/othe\
r-sites.html      248     -       abcdefgh.ijk.EDU        -
```

Recently, archie logs were recently added to the system. The new code to do this took less than 30 minutes and was easily added. The query was placed in the *name* field, which, with hindsight, should probably be described as a *request*. Like described above, since there was no *response* / reply / status field, the number of hits returned was just appended to the operation field:

```
archie  query/matches=19/esttime=40     1994-11-01-01:06:10     wnbff2\
0b.zip    -       nobody  123.456.789.01  -
```

Conclusions

A flexible and efficient combined log system has been designed and implemented. It automatically processes active log files being written concurrently by software daemons, stores the collected information in a readily accessible format and provides summaries for users. In addition, it is easily customisized and the data generated is easy to access by programs outside the system, since each line has an easy-to-use format that well known programs like grep, awk, sed and wc can process.

If you wish to obtain and try out this software, it can be found at the HENSA Unix[3] archive by WWW[4], ftp[5] or email[6].

Thanks go to the HENSA Unix staff: Maggie Bowman, Tim Hopkins and Neil Smith for their help in designing of this software and for looking over much earlier drafts of this paper as well to the anonymous reviewers for their useful comments.

References

[conolly95]
Daniel W Conolly: Public Text of the HTML 2.0 Specification, 1995, <URL:http://www.hal.com/users/connolly/html-spec/>

[fielding94]
Roy Fielding: wwwstat, processes only NCSA WWW logs, March 1994, <URL:http://www.ics.uci.edu/WebSoft/wwwstat/>

[hughes94]
Kevin Hughes: getstats, processes gopher plus CERN, NCSA, Plexus, GN and common WWW logs, February 1994, <URL:http://www.eit.com/software/getstats/getstats.html> and <URL:ftp://ftp.eit.com/pub/web.software/getstats/>

[magid94]
Jonathan Magid: fwgstat, processes FTP, Gopher, WAIS and the NCSA and Plexus HTTP logs, 1994, <URL:ftp://ftp.sunet.se/pub/archiving/ftp/fwgstat-0.035.shar>

[myers94]
Chris Myers: xferstats, processes only FTP logs and available as part of the Wuarchive FTP daemon software, 1994,

<URL:ftp://unix.hensa.ac.uk/pub/walnut.creek/FreeBSD/FreeBSD-current/ports/net/wu-ftpd/ util/xferstats>

Footnotes

[1]

This work was done with funding from COMETT for transputer and occam training and the JISC SEL-HPC project.

[2]

Parallel Computing Archive at HENSA Unix - <URL:http://www.hensa.ac.uk/parallel/>.

[3]

HENSA Unix Archive - <URL:http://www.hensa.ac.uk/>.

[4]

Combined Log Tools by WWW - <URL:http://www.hensa.ac.uk/tools/www/logtools/>

[5]

Combined Log Tools by ftp - <URL:ftp://unix.hensa.ac.uk/tools/www/logtools/>/>

[6]

Combined Log Tools by sending an email message to `archive@unix.hensa.ac.uk` with the contents: `send /tools/www/logtools/README` or `help` for more information.