PHHTTPD

Zach Brown

PHHTTPD
by Zach Brown

Table of Contents

Chapter 1. Introduction

phhttpd is an HTTP accelerator. It serves fast static HTTP fetches from a local file-system and passes slower dynamic requests back to a waiting server. It features a lean networking I/O core and an aggressive content cache that help it perform its job efficiently.

1.1. Architectural Overview

phhttpd features a very slim I/O core. It does all its networking work using non-blocking system calls driven by whatever event model is most appropriate for the host operating system. This allows a single execution context to handle as many client connections as the event model dictates.

phhttpd's job is to serve static content as quickly as it possibly can. To do this it maintains a cache of content in memory. When a request is serviced, phhttpd saves a reference to the on disk content and whatever HTTP headers are dependent on the content. Next time a request for this content is received, phhttpd can service it very quickly. This cache can be prepopulated-populated at run time, or can be built dynamically as requests come in. Its size may also be capped by the administrator so that it doesn't overwhelm a system.

phhttpd is a threaded stand alone daemon. The number of threads is currently statically defined at run time. Incoming connections are evenly balanced among the running threads, regardless of what content they may be serving. Connections are served by the thread that accepted them until the transfer is done.

1.2. Supported Systems

phhttpd is currently only expected to build and run on Linux systems using glibc2.1 under a kernel that supports passing POLL* information over real-time

SIGIO signals. This means later 2.3.x kernels or a 2.2.x kernel that has been patched.

I badly want this to change. If you're interested in doing porting work to other Operating Systems, please do let me know.

Chapter 2. Configuration File

2.1. Overview

phhttpd uses an XML config file format to express how it should behave while running. More information on XML may be found near http://www.w3.org/XML/

phhttpd's configuration centers around the concept of virtual servers. For us, a virtual server may be thought of as the merging of a document tree and the actions phhttpd takes while serving that content.

phhttpd.conf may be thought of as having two main sections. The global section, which defines properties that are consistent across the entire running phhttpd server, and multiple virtual sections that describe properties of that only apply to a virtual server. There will only be one global section while multiple virtual sections are allowed.

2.2. Global config section

The global section defines properties of the running server that don't apply to a single virtual server. It should be enclosed in

Global config entities

cache max=NUM

Sets the maximum number of cached responses that will be held in memory. Each cached responses holds a minimal amount of memory. More importantly, each cached response holds an open file descriptor to the file with real content and an mmap()ed region of that content. phhttpd will start pruning the cache when it notices either of these two resources coming under pressure, but has no way to easily deduce that its running low on memory. The administrator may set this value to set an upper bound on the number of responses to keep in memory.

control file=PATH

This specifies the file that will be used to talk with phhttpd_ctl.

globallog file=PATH

This specifies the file to which global messages will be logged.

mime file=PATH

This specifies the file that contains the mapping of file extensions to MIME types. It should be of the form:

```
text/sgml        sgml sgm
video/mpeg       mpeg mpg mpe
```

timeout inactivity=NUM

Controls various network connection timeouts. 'inactivity' sets the amount of time that a connection can be idle before phhttpd will forcibly disconnect it. inactivity defaults to 0, which lets the connections idle until TCP timeouts take effect.

sendfile

Enabling this option tells phhttpd to use sendfile() rather than write()ing from an mmap()ed region. Avoiding calling mmap() will shorten the amount of time it takes to build cached responses.

2.3. Virtual Servers

A Virtual Server can be thought of the abstraction of serving up a content tree ("docroot" in apache speak). There are a set of attributes that are used to define a virtual server. These attributes are used to decide which virtual server will process a client's request. Then there are attributes which define how the content is served.

A virtual server must have a docroot. The virtual tag in the config file has a docroot attribute that must be set.

```
<virtual docroot=PATH>
...
</virtual>
```

There can be as many virtual sections in the configuration file as one likes.

Global config entities

md5

This enables the generation of the Content-MD5: header. This greatly increases the cost of creating a cached response for this virtual, because the MD5 function must be applied to the entire content of the response. Once the response is created, though, there is no per-request overhead.

prepop

This will cause phhttpd to traverse the entire docroot at initialization time and prepare cached responses for all the files it finds. This happens in the back ground during normal operation, so there is no dramatic increase in the time it takes for phhttpd to start serving connections.

name

This tag surrounds the string that will be used to identify the server. This string will be compared to the Host: header given in the request from the client, or will be compared to the 'host part' of the full URL if that was given. This will be used in combination with the network address and port pair to determine if a request should be served by a virtual server.

listen v4=DOT.TED.QU.AD port=PORT

This virtual server will be chosen to serve an incoming request if that request was made to the network address specified in this entity. There can be as many of these as one likes in a given virtual server, and '*' may be specified for either parameter to indicate that all addresses or ports should match.

logs

The logs section of the virtual server define the per virtual log files that should be written to during operation. See the following section on logging.

Chapter 3. Logging

“All kids love log!”

3.1. Overview

phhttpd maintains log buffers for each log it writes too. Logged events are put in these buffers at reporting time rather than being immediately written to disk. These logs are written as they are filled during normal operation, or at regular intervals. This greatly reduces the performance impact of keeping detailed logs.

3.2. Configuration

phhttpd keeps interesting logs on a virtual server granularity. The action of recording lots is specified by including an entity in the log section of a virtual for the log source that wants to be kept. There is an entity for each source of logging, and attributes to that entity define where it is logged to. It looks something like this:

```
<logs>
<LOGSOURCE mode=OCTALMODE file=PATH>
...
</logs>
```

mode is the octal permissions mode of the file that is to be opened. As it is parsed by dumb routines, a leading 0 is highly recommended. fileis the file the logged events will be written to. The LOG_SOURCE is one of:

access	Successfully answered requests

agent	The value given in the 'User-Agent' HTTP request header
referer	The string given in the 'Referer' HTTP request header

3.3. Format and Strange Behaviour

phhttpd log entries are contained with a single line in a text file. They contain the time the log entry was written, an opaque token that is associated with the connection that caused the log entry, followed by the actual entry.

The contents of the 'referer' and 'agent' log entries is simply the string that was given with the header. The contents of the 'access' log is a little more interesting. It has the decoded relative URL that was asked for, followed by the total bytes that were transfered, and the time in seconds that it took to transfer.

```
387f7a45 387f7a45800210ac8910500 /index.html - 2132 0
```

is an entry from an 'access' log.

The first field is the time in seconds since the Unix epoch, a.k.a. time_t. The second field is associated with the client connection that caused the log entry. It is constant for the duration of the connection, and is written to all the logs entries, of whatever type, that are generated. This allows a log parser to do more complete connection granularity analysis. As it happens, this opaque token is currently built up of the time the client was connected, its remote and local network address, etc, but these values most _not_ be parsed as they may change in the future.

Entries generated by a thread will be written in chronological order. If, however, multiple threads are sharing an output file the resulting entries may not be written in chronological order. It is up to the parsing programs to use the 'time' field to sort by, if they care about chronological order.

Chapter 4. Run Time Facilities

4.1. Overview

While phhttpd is running it listens to a 'control' socket for messages from the administrator. The currently provided phhttpd_ctl program allows the administrator to minimally interact with phhttpd. This provides both control and status reporting.

phhttpd_ctl always wants a –control argument that specifies the control socket of the running phhttpd daemon. This should match the <control> tag specified in the config file.

4.2. Log Rotating

phhttpd can be told to rotate its logs so that existing logs may be processed.

The –rotate argument to phhttpd_ctl tells phhttpd to rename the existing files to a unique name, open new files with the previously used names, then close the renamed logs and start using the newly created files. phhttpd_ctl will output the names of the newly created files which will be safe to use once the command exits.

The –reopen argument to phhttpd_ctl tells phhttpd to close the existing file logs and reopen the files with the filenames that were configured. This implies that an external entity has moved the files to new names and wants phhttpd to stop using them.

4.3. Status Reporting

The –status argument to phhttpd_ctl tells phhttpd to return a quick status blurb about the server. It contains miscellaneous information about the running state of the server.