Software Architecture Case Study Report for Apache HTTP Server

Ashwini Tokekar [201405546],Deepak Kathayat [201101213],M.S.Soumya [201450872]

May 2, 2016

Contents

1 INTRODUCTION **2**
- 1.1 History of Apache . . . 2
- 1.2 Uses . . . 2

2 REQUIREMENTS AND QUALITIES **2**
- 2.1 Feature overview . . . 2
- 2.2 Functional Requirements . . . 3
- 2.3 Non-Functional Requirements . . . 3
- 2.4 Functionalities provided by Apache HTTP server . . . 3

3 ARCHITECTURAL SOLUTION **4**
- 3.1 Overview . . . 4
- 3.2 Two-Phase Operation . . . 4
  - 3.2.1 Start-up Phase . . . 5
  - 3.2.2 Operational Phase . . . 6
  - 3.2.3 Shutdown . . . 6
- 3.3 Multi-Processing Modules . . . 7
- 3.4 The leader-followers pattern . . . 7
- 3.5 Preforking Architecture . . . 8

4 REFERENCES **10**

5 EFFORT **11**

1 INTRODUCTION

1.1 History of Apache

The Apache HTTP server is an open source project which was built collaboratively by 8 contributors often known as the core contributors. The software development was aimed at creating a robust, commercial-grade, and featureful implementation of an HTTP server. Now the organization has evolved and has a large number of committers who are given access to the source code of the repositories to help maintain the project or its relevant docs. The core group now manages the project and is called the Project management Committee (PMC for short). The name Apache was chosen as a sign of respect for the native American Indian tribe of Apache, well known for their superior skills in warfare strategy and inexhaustible endurance. It also makes a cute pun on "a patchy web server" – a server made from a series of patches. The group of developers who released this new software soon started to call themselves the "Apache Group".

1.2 Uses

The apache web server is used for hosting web sites.

2 REQUIREMENTS AND QUALITIES

2.1 Feature overview

Apache supports a variety of features, many implementations compiled as modules that extends the basic functions. These can range from support for server-side authentication system programming language. Some common language interfaces support Perl, Python, Tcl, and PHP. Modules include popular approval $\text{mod}_{\text{access}}$, the mod_{auth}, $\text{mod}_{\text{digest}}$ and $\text{mod}_{\text{authdigest}}$ successor to $\text{mod}_{\text{digest}}$. Other features examples include Secure Socket Layer and Transport Layer Security support (mod_{ssl}), the proxy module ($\text{mod}_{\text{proxy}}$), the unit re-write the title ($\text{mod}_{\text{rewrite}}$), the special log file ($\text{mod}_{\text{logconfig}}$), and support for the candidate ($\text{mod}_{\text{include}}$ responsible and $\text{mod}_{\text{extfilter}}$).

Some of Apache¡¯s popular compression methods, including external expansion modules, mod_{gzip} are implemented to help reduce the size (weight) of web pages via HTTP. ModSecurity is an open source Web intrusion detection and prevention of the engine application. Apache logs can be analysed using free scripts such as AWStats software / W3Perl or Visitors through a Web browser.

Virtual hosting allows an Apache installation to serve many different sites. For example, single installation can serve www.example.com, www.example.org, test47.test-server.example.edu etc. simulateonusly.

Apache has error messages that are configurable, with the authentication databases formality-based database management systems, and allows the content negotiation. And also it supports by many of the graphical user interface (GUI) for. It supports password authentication and digital certificate authentication. Since the source code is free, anyone can be adapt the server to their specific needs, also there is a library of Apache plug-ins.

2.2 Functional Requirements

- URI to filename translation
- Several phases involved with access control
- Determining the MIME type of the requested entity
- Actually sending data back to the
- Client logging the request.

2.3 Non-Functional Requirements

- Per-directory configuration
- Allowing native functionality to be superseded
- Backward compatibility ensured
- There must be protocol independence

2.4 Functionalities provided by Apache HTTP server

HTTP server and proxy features Loadable Dynamic Modules Multiple Request Processing modes (MPMs) including Event-based/Async, Threaded and Prefork. Highly scalable (easily handles more than 10,000 simultaneous connections) Handling of static files, index files, auto-indexing and content negotiation .htaccess support Reverse proxy with caching Load balancing with in-band health checks Multiple load balancing mechanisms Fault tolerance and Failover with automatic recovery WebSocket, FastCGI, SCGI, AJP and uWSGI support with caching Dynamic configuration TLS/SSL with SNI

and OCSP stapling support, via OpenSSL. Name- and IP address-based virtual servers IPv6-compatible HTTP/2 protocol support Fine-grained authentication and authorization access control gzip compression and decompression URL rewriting Headers and content rewriting Custom logging with rotation Concurrent connection limiting Request processing rate limiting Bandwidth throttling Server Side Includes IP address-based geolocation User and Session tracking WebDAV Supports Embedded Perl, PHP and Lua scripting and CGI $\text{public}_{\text{html}}$ per-user web-pages Generic expression parser Real-time status views XML support

3 ARCHITECTURAL SOLUTION

3.1 Overview

Apache HTTP server contains a relatively small core, together with a number of units. The units can be statically compiled into the server or, more commonly, stored in the / modules / or the *libexec* guide maintain and then can be dynamically loaded at runtime.

In addition, the server relies on the Apache Portable Runtime (APR) libraries, which provide a cross-platform operating system layer and utilities,so that modules don¡¯t have to rely on non-portable operating system calls.In addition, the server depends on the Apache Portable Runtime (APR) libraries that provide a cross platform operating system layer and utilities, so that the units do not have to rely on the operating system calls that are not portable.A module for special purposes, the Multi-Processing Module (MPM), is working to improve the Apache to the underlying operating system. It must be the MPM that usually is the only unit to gain access to another operating system through APR.

3.2 Two-Phase Operation

Apache Server operates in two stages: start-up and operational. At Start-up stage the server is in root mode, and includes analysis of the configuration file(s), loading modules, and the instantiation of system resources such as log files, shared memory segments, and database connections. In Normal operation, Apache gives up the system privileges and works as a normal user before accepting and processing connections from clients across the network. This two-stage process has some implications for applications architecture.

First, you must run anything that requires system privileges in the startup. Second, it's a good practice to run as much as possible configu-

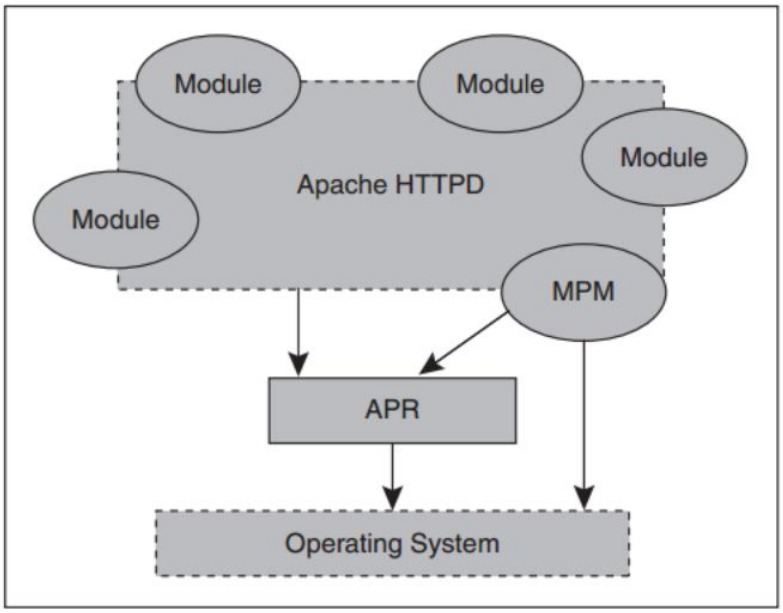

Figure 1: Overview diagram

ration at startup, to ensure that processing is reduced when each request is serviced.On the contrary, because a lot of slow and expensive operations are concentrated in the startup stage, it would be inefficient to try run the Apache from a generic server, such as inetd or tcpserver. A non-trivial aspect of the architecture is that the code configuration is, in fact, executed twice at startup (although not at the restart).The first time through the checks that the configuration is valid (at least to the point that can successfully start Apache); the second pass is "live" and leads to the operational phase. Most units can be ignore this behavior (the use of the standard in APR pools ensure that it does not cause a resource leak), but may have an impact on some units.For example, the unit that holds the new code dynamically loads it in the startup phase may want to do this only once, and therefore, it must use a technique such as setting and checking of a static flag to ensure it¡¯s creation of one-time only.

3.2.1 Start-up Phase

The purpose of the startup of the Apache is to read the configuration, load modules, libraries, and create the necessary resources. Each unit may have

its own resources, and has an opportunity to create those resources. In the startup phase, Apache runs as a single process,single-thread program and has root privileges.

- Configuration
  The main configuration file in Apache is usually called as httpd.conf file. However, this is just a convention, a third-party Apache distribution such as those provided as .rpm or .deb packages may use different nomenclature.In addition, it may be a single file, or may be divided into multiple files using the Include guidance to include various configuration files. Some distributions have a very complex configurations.For example, Debian GNU/Linux Apache configuration, relies heavily on the familiarity with Debian, not with Apache. The configuration file httpd.conf file is a plain text file which is analysed line by line at server startup. The contents of the file include directives, containers, and comments. It also allows empty lines and leading whitespaces, but will be ignored.

3.2.2 Operational Phase

At the end of the start-up stage,the control passes to the Multi-Processing Module(MPM). The MPM is responsible for managing the Apache functioning on the systems level. It usually does so by maintaining a set of worker processes and/or threads, as appropriate with the operating system and other applicable constraints (such as the optimization for a specific use).The original process remains as the "master" to maintain a set of worker children. These workers are responsible for entertaining incoming connections, while the parent process deals with the creation of a new children , and the removal of the surplus when necessary, and communication signals such as "shutdown" or "restart."Because of the MPM structure, it is not possible to describe the operation phase in the definite terms. The standard MPMs use the worker children in some manner so that, they are not limited only work in one way. Thus, it can be another MPM achieve in principle, to implement a completely different system-level server architecture.

3.2.3 Shutdown

There is no shutdown phase as such. Instead, anything that needs be done on shutdown is registered as a cleanup. When Apache stops, all registered cleanups are run.

3.3 Multi-Processing Modules

At the end of the start-up phase, having been read configuration, full control of the Apache passes to a Multi-Processing Module. MPM acts an interface between the running Apache server and the underlying operating system. Its primary role is to improve the Apache platform, while ensuring the server to work efficiently and securely. As the name suggests, MPM is in itself a module . But MPM is a unique systems level (even the development of MPM is outside the scope of a book about the development of applications). Also uniquely, Apache instance contains single MPM, which is determined at build-time. An MPM¡¯s responsibilites are located inside the main loops of the Apache. The main server does the initialization and configuration for processing before calling $ap_{mpmrun}()$ function in main() to be delivered to MPM.It is the responsibility of the MPM to take care of starting the threads and / or processes as required. MPM will also be responsible for listening on sockets for incoming requests. Upon the arrival of requests, the MPM distributes them between the threads and/or processes that have been created. And then run the Apache standard procedures for request handling. When you restart or shut down, the MPM delivered back to the main server. So,each server functionality is still the same for any MPM, but the multi-processing model interchangeable. The figure below shows the responsibility of Apache MPM in the general behavior of the Apache. The dotted line represents the actions of MPM takes responsibility.

3.4 The leader-followers pattern

The preforking structure is based on a set of tasks (processes or threads) that play three different roles:

- Wait for requests (the listener)
- Request processing (worker)
- Queue in wait to become a listener (idle worker)

Listener is the leader.The right to wait for connection requests can be granted to only one task. If listener gets a request, it provides the right to listen, he switches the role to worker, which implies that it processes the request using the connection established as listener. If it is done processing the request, it will close the connection becomes idle worker. This means that it queues in waiting to become the listener. Usually an idle worker task will be suspended.

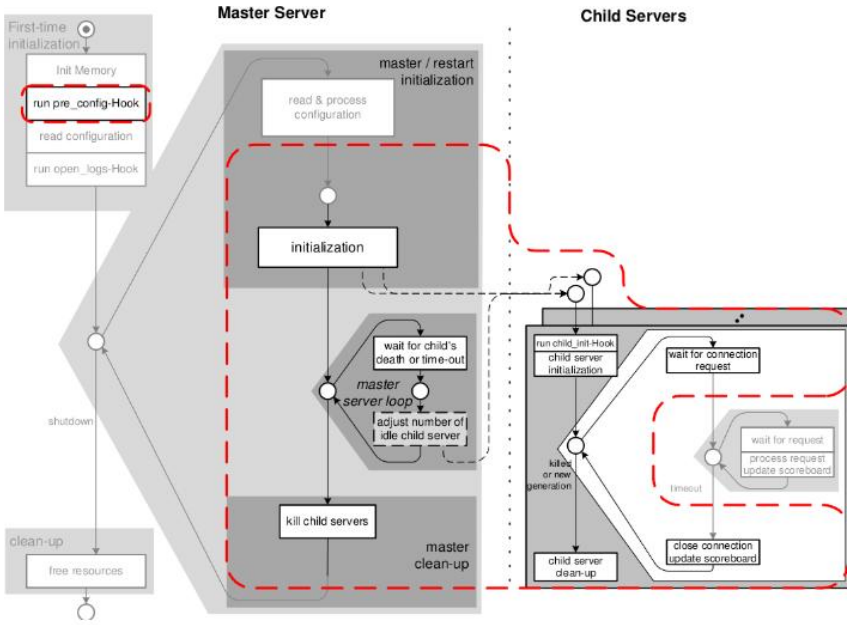

Figure 2: Responsibility of an Apache 2.0 MPM

3.5 Preforking Architecture

The the first multi-tasking architecture of Apache was Preforking Architecture. In Apache 2.0, which is still the default MPM for Unix. The Netware MPM is very similar to the Apache Preforking except that it uses Netware threads instead of Unix processes.

Summarizing, the Apache Preforking Architecture takes the traditional approach where each child server is a process in itself. This makes Preforking stable architecture but also reduces performance. The following figure shows how the Apache Web Server provides a powerful multi-processing module, which is based on preforking process, Apache-MPM prefork.

The following figure shows the overall behavior of the server, including the master server and the child servers.

Independent of the multitasking architecture, Apache¡¯s behavior consists of the sequence of following parts that will be discussed individually for each of the architecture:

The first time initialization Resource allocation, and read and review configuration, become a daemon.

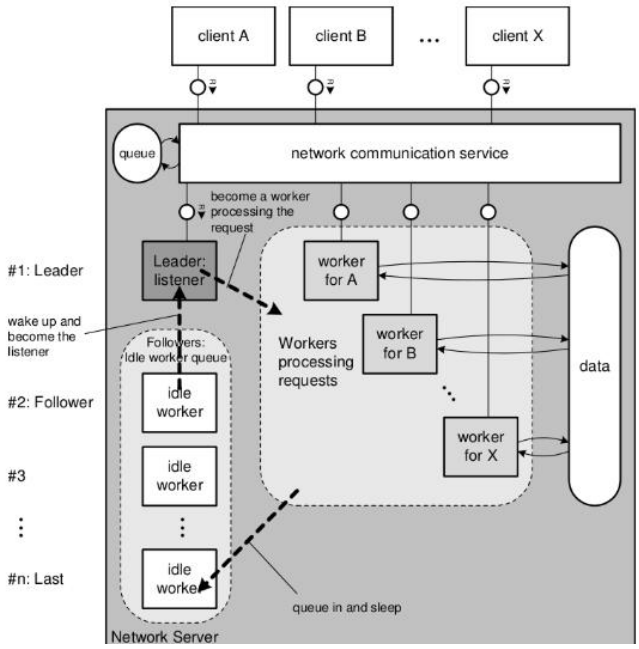

Figure 3: The leader-followers pattern used in the preforking server architecture

The restart loop Re-read configuration, create task pool gathered from the child server processes and enter the main server loop.

The main server loop To control the number of idle child server processes in the job pool.

Respond to a request loop (only child server) Wait to become a leader, and wait for a connection request, become a worker and enter the keep-alive loop.

The keep-alive-loop(only child server)

- Handle HTTP requests
- Cleaning master server and child servers before deactivation

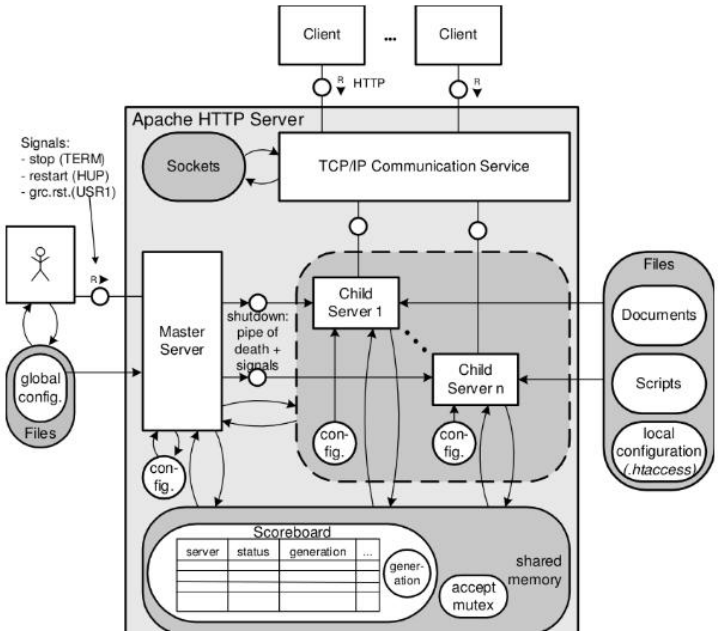

Figure 4: The Apache 2.0 Preforking MPM

4 REFERENCES

References

[1] Amy Brown and Greg Wilson. *The Architecture of Open Source Application- Elegance, Evolution, and a Few Fearless Hacks*,

[2] *https://httpd.apache.org/ABOUT*$_A$*PACHE.html*.

[3] *http://berb.github.io/diploma-thesis/original/042*$_s$*erverarch.html*.

[4] *http://www.fmc-modeling.org/category/projects/apache/amp/4*$_3$*Multitasking*$_s$*erver.htmlsub* : $Overview - Apache - Multitasking - Arch$.

[5] *http://ptgmedia.pearsoncmg.com/images/9780132409674/samplechapter/kew*$_c$*h*02*.pdf*.

[6] *http://apachecon.com/2007/notes/t02-notes.pdf*.

[7] *http://iw3c2.cs.ust.hk/WWW5/www5conf.inria.fr/fich*$_h$*tml/papers/P*20*/Overview.html*.

[8] *http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=612229*.

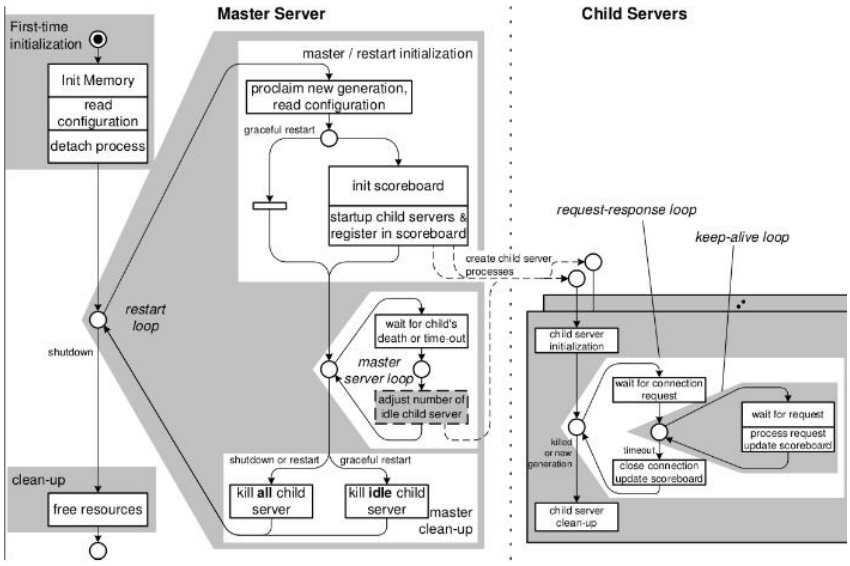

Figure 5: Overview: The behavior of Apache

5 EFFORT

Name	Time spent
Ashwini	7 hrs
Deepak	7 hrs
Soumya	7 hrs