Web routers: An explorative definition

Sdf

Richard (Rikki) Andrew Cattermole

2016

Abcd

# Introduction

Since the inception of the World Wide Web (www) in 1989 there has been a large uptake in its usage by everyone all across the globe at an estimate of 3.4 billion users as of July 1st of 2016. With every one of those users working with the standards of Uniform Remote Locator, Hypertext Transfer protocol and Hypertext Markup Language in some form or another.

TODO: /\ reference <http://www.internetlivestats.com/internet-users/>

TODO: /\ reference [https://www.w3.org/Help/#invention](https://www.w3.org/Help/" \l "invention)

TODO: /\ reference <https://tools.ietf.org/html/rfc3986>

TODO: /\ reference <https://www.w3.org/MarkUp/draft-ietf-iiir-html-01>

During the early days many different web browsers and servers were created. Majority of these have since long died off but the legacy that is the definition of each has not. These definitions are:

TODO: /\ reference <https://www.w3.org/wiki/How_does_the_Internet_work>

* Web browser  
  A program to which allows display and execution of a web page for a user. Interacts with a web server to provide any data required.
* Web server  
  Retrieves files or resources from the file system or some form of backend such as a web application and send them to the client as requested.
* Web API/service  
  A standalone piece of software that will dynamically create content to send to a client.

With an upsurge in internet speeds during 1990s as demonstrated by Nielsen’s Law, companies and developers alike experimented with dynamic web pages allowing for user interactions not possible with static web pages alone. The Common Gateway Interface (CGI) was created to allow for external program to be executed as part as the web page processing. From this point on existing programming languages gained uses that was not seen before which helped to introduce other new programming languages. An example of a new programming language spawned by this would be PHP.

TODO: /\ reference https://www.nngroup.com/articles/law-of-bandwidth/

TODO: /\ reference <https://tools.ietf.org/html/rfc3875>

TODO: /\ reference <http://php.net/manual/en/history.php.php>

TODO: /\ another example of language spawned e.g. perl?

Web servers and web (server side) APIs alike are a field of research that continues to introduce new areas of study for research in both a formal and an informal capacity. Combined they share a very similar technology set only with slightly different purposes and entry points. Of which the router that resides on a server from which all fundamental mental models originates is conceived.

TODO: https://www.w3.org/wiki/How\_does\_the\_Internet\_work

A web routers primary goal is to map any incoming request to a function to process it. The execution and processing of a request once mapped can be done in any number of languages and quite commonly utilities other protocols such as a Fast-CGI to communicate to another process to execute it.

TODO: drop fast-cgi here

TODO: diagram of [client] send request [server] listen→http process → route → handle → respond

As most users consider primarily the usage criteria of a web service framework (utility code for database interaction and templates) instead of the implementation, there are little consideration by its users for the implementation and along with it, its performance. Because of little amount of thought that goes into replacing it or to get it better match it with the usage patterns there are performance penalties incurred by most implementation in some form or another. The making of these algorithms and data structures to sufficiently support multi-threading in any reasonable manner is a difficult task as shown by the Computer Science literature into the subject of data structures.

TODO: research into data structures + multi-threading for performance penalities

Web servers are a highly concurrent piece of software and require high levels of performance at the risk of requiring more servers to handle requests which can have substantial costs accredited to it, yet some crucial parts of it have limited formal research done onto it which could potentially pose an unknown quantity to end latency to an HyperText Transfer Protocol request. To get more performance out of a website there are two methods, more hardware thrown out of it or change algorithms to make it more efficient. The focus of this research is towards the software as more hardware is quite expensive and is already well understood.

TODO: reference HTTP spec about where it lies in terms of requests

TODO: reference some kind of general overview of scaling websites

The potential unknown nature that is request routing has got to have declaration into if and by how much the cost of it is incurred per request. This research will focus upon how much does a request cost depending upon the data structure and hence implementation involved in a set scenario.

# Literature review

At the core of a web router is the process of turning a request from Hyper Text Transfer Protocol (HTTP) 1.0/1.1/2.0 into a call to procedure by which handles the request and returns the result along with some meta information such as cache information. This process has several stages:

1. Socket listening
2. HTTP request processing
3. Routing to function call
4. HTTP response creation

The above list is a general overview of the different sequential parts that a request goes through on the server. Commonly it is separated out in for the implementation into:

1. Asynchronous socket listener
2. Thread/Fiber router (choose the thread to execute the request handling in)
3. HTTP request processing
4. Routing to function call
5. HTTP response creation and return

However this does not take into account the need for blocking actions such as file reading/writing and other socket related processing such as database access. So for this use case to get the best performance the usage of Fibers is commonly used. Fibers allow for using up as much of the time slice that the kernel is willing to give a thread while also switching out what code (and with it its stack) is being executed at any given moment. Improving the number of requests a single thread is capable of execution before its time slice is ended or the blocking operations takes precedent.

Asynchronous execution along with threads and fiber handling is a complex topic that can affect performance between web servers quite significantly. This can be seen between Nginx and Apache2 httpd. Nginx uses asynchronous event based handling while Apache2 utilizes a thread based approach.

TODO: reference <https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations> or <http://blog.smartbear.com/sqc/nginx-the-faster-web-server-alternative/>

When a connection has been established and the handling code is available to being execution, it will translate the given binary stream into some form of programmatic representation such as classes or structs. Allowing for ease of use and modification by the routing engine. The handling code may be a wrapper to another protocol such as Fast-CGI to allow out of process execution and processing of the request.

The routing engine is responsible for manipulating this request representation into recognizing a specific route. This is primarily done by utilizing some kind of run-time look up and registration into the routing table. This allows for using language features such as attributes to map procedures to routes more organically. For example the web server Nginx, utilizes Red-Black trees for files caches and Fast-CGI processing. From this the handling mechanism for the specific route is called with it.

TODO: reference <http://httpd.apache.org/docs/2.4/developer/request.html>

TODO: reference <https://trac.nginx.org/nginx/browser/nginx/src/http/modules/ngx_http_fastcgi_module.c?rev=953512ca02c6f63b4fcbbc3e10d0d9835896bf99>

TODO: reference https://trac.nginx.org/nginx/browser/nginx/src/http/ngx\_http\_file\_cache.c?rev=953512ca02c6f63b4fcbbc3e10d0d9835896bf99

## The Hyper Text Transfer protocol

TODO: Basic what is it?

TODO: overview of basics of each version \/

Versions:

1. 1.0
2. 1.1
3. 2.0

TODO: why was 2.0 needed? Problem + its solutions

TODO: what does a request provide?

TODO: what does a response contain?

TODO: overview of how requests work in terms of sockets (need a nice pretty image)

## Optimization techniques

TODO: cache locality

TODO: /\ what do databases do?

TODO: memory layouts a reference for this: http://www.sciencedirect.com.ezproxy.lincoln.ac.nz/science/article/pii/S2214579615000635

TODO: /\ what do databases do?

TODO: JIT’ing

TODO: adding threads

TODO: /\ what do databases do?

TODO: more efficient instructions

TODO: less work

TODO: rearranging (e.g. one whole regex vs a whole lot of parts)

## The routing problem

By using the definition of a router as the process to which the decision of which route handler is chosen per request and along with it the definition of what the routing problem is, it can be seen that this can be implemented in many different and varying ways. Because of this the approaches can differ quite significantly. These different approaches each have a different set of costs and cannot be interchanged in a given context with the expectation of performance related problems appearing.

TODO: some kind of official documentation to point towards what a router is

TODO: footnotes showing some existing routers e.g. https://github.com/klein/klein.php

Not all routers have the same feature sets supported. But at the core of it they are just a way to take in a set of known variables and return a function to execute. These set of variables that must be utilized in each searching of the underlying structure is fairly unique compare to the research into data structures in Computer Science which focus primarily upon a single value. With the extension of multiple variables to check and a more complex search algorithm that may need to repeat itself the usage of existing data structures and algorithms may be used.

TODO: reference basic router design!!!!!!!!!

Most implementations only work with the path from the HTTP header and because of this require the least amount of extensions to existing data structures and algorithms. This allows them to be implemented using only regex or other more limited approaches. This does cover most cases; by utilizing multiple instances it can be used for different HTTP methods such as GET and POST. However other rules such as rewriting are not possible and this is where tree graphs become quite useful.

TODO: regex router footnote

TODO: router that uses GET/POST footnote

When it comes to other than regex approaches, generally quite a bit more information is stored associated with a single router handler function. This can be done by using another data structure as a key in a map or to wrap the reference to the handler function. This allows it to use other conditions such as the HTTP request fields of User-Agent, Referer or Host. Support of this is a significant complexity increase and results in limited formal research that was discovered in the creation of this proposal.

TODO: something, ANYTHING on unconditional (my term) condition nature of data structures in CS, include about threading and how locking can affect this along with memory usage also cache locality of memory

TODO: ok this is a huge amount of text that is needed here

## Existing implementations

TODO: Apache 2 web server

TODO: Nginx

TODO: some comparison and referencing?

## Existing data structures

TODO: lists

TODO: maps

TODO: tree graphs and more?

# Research question

# Method

TODO: ok we got a methodology, but a) what is it b) why is it that c) what does that give us?

## Code

TODO: so an outcome is code, what does the code show and how are we doing it?

### Router types (known)

TODO: some router types are already known, so lets define them as part of the scope of the project

# Time line

TODO: tasks and explanation

TODO: do it based upon the tasks that need to be completed and estimate how long each one will take

# Budget

It is expected that a printing budget of $100NZD will be needed for final copies available for submission.

No other expenses has been expected to be incounted.

# Outcomes

There is one outcome expected from this proposal and that is the dissertation. Supporting this will be code created to exemplify the different kinds of web routers that currently exist today.

TODO: Think of the outcome in terms ok new knowledge rather than artefacts

TODO: the above is absolutely horrible but there needs to be something along those lines

TODO: there also needs to be something else here, what could that be?

## Dissertation

TODO: yes there is a dissertation with a methodology but what else can I do/need?

# References

TODO: do we need any more?

TODO: http 2.0 spec reference

TODO: asynchronous socket libraries

TODO: apache2 mod\_rewrite

TODO: fastcgi

TODO: PHP (webserver→PHP routing)

TODO: cache locality optimization strategies for data structures and general memory such as arrays