Web routers: An explorative definition

Sdf

Richard (Rikki) Andrew Cattermole

2016

Abcd

# Introduction

Web servers and web applications alike are an emerging field of research in the world of Information and Technology (IT). They share very similar technology only with slightly different purposes and entry points. Of which the router from which all fundamental mental models originates from is convceived.

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 fast cgi reference

The execution of a request handling is quite an extensive place of research with many frameworks (ASP.net, Play Framework, JSP and Zend) being used in a vastly differing environments to each other but fundamentally doing the exact same things. Primarily this is 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. In the end the usage of the router itself is not explicit in function calls by users instead relying on the magic that is the framework and run-time look ups to hook everything up without understanding what is going on.

TODO: foot-notes for ASP.NET, Play Framework, JSP and Zend

As most users do not explicitly work with the router when considering frameworks, there is little consideration for its implementation and along with it, its performance. Because very little thought goes into replacing it or to get match better 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

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.

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

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 into a procedure by which to handle and return result along with some meta information such as cache information. This is done by separate parts:

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

The above list is a general overview and problems that an implementation may face. Commonly it is instead separated out 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 will not be further discussed in this literature review. These topics alone are worth research

/\ TODO: see (a) (b) (c) for further information on these matters.

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 such as what is done with Apache Web Server version 2’s mod\_rewrite. The given procedure that goes on to execute the code for that route takes in the request representation along with a response representation ready to send the HTTP response representation to the client. The exact nature of the execution handler does not matter. It could be PHP or a web service framework written in the same language calling into its own router.

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

TODO: JIT’ing

TODO: adding threads

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.

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?

# 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: 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?

### Methodology

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

# 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