Web routers: An explorative definition

Sdf

Richard (Rikki) Andrew Cattermole

2016

Abcd

# Introduction

Web routers are an ill-defined central component within every web server and along with it service or application. The full feature set is an unknown quantity for which performance penalties could be numerous especially in higher level languages such as Java or JavaScript. The designs are core and based upon primal structural algorithms such as lists and tree graphs but with an added problematic twist of run time described limitations and mapping of the requests.

Work has been done to attempt the definition of this component but it has mostly been informal. This is a problematic situation for Academics and for those who care about best practice in the world of IT.

TODO: more

This dissertation proposal brings forth the desire and willingness to describe the component in terms that can be implemented and explored for future generations.

TODO: what goes here?

# Literary 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 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)

## The routing problem

Routers can be implemented using a variety of different ways. However the approaches differ significantly enough that feature sets and performance may not compare between them for any given set of data (routes + requests). This may be further exemplified by looking at the approach taken at searching for elements in lists and maps. However these primal data structures are not what is commonly used, instead the usage of tree graphs is preferred. Not because it is inherently better but instead it allows for skipping large sections or sub parts of a set of requests. Narrowing down the search for the specific request significantly more faster.

Most implementations only work with the path from the HTTP header. This allows them to be implemented using only regex or other more limited approaches such as using a map. 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.

Unlike the approach that regex typically takes, an extra data structure provided with the route function can be provided to allow for more complex operations such as rewriting or basing it off of other fields such as User-Agent or Referer. This can be used with lists, maps or tree graphs. Because of the extra information a router has access to and can query in its data store it has potentially many conditional arguments per search, without the possibility of rewriting of a request. This is a unique situation in computer science, the existing literature on the subject is thin as more commonly you are searching for a specific value instead of many potential points that need interpreting while searching.

TODO: something, ANYTHING on unconditional 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

# 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