**Research Proposal**

**PhD/Masters/Honours**

|  |  |
| --- | --- |
|  | |
| Faculty: | **Environment, Society and Design** |
|  | |
| Department/School: |  |
|  | |
| Degree: | **B.Science(Honours)** |
|  | |

|  |  |
| --- | --- |
|  | |
| Student Name: | **Richard (Rikki) Andrew Cattermole** |
|  | |
| Student ID Number: |  |
|  | |
| Address: | **52 Bethel Crescent**  **Christchurch**  **New Zealand** |
|  | |
| Email Address: | **alphaglosined@gmail.com** |
|  | |

|  |  |  |
| --- | --- | --- |
|  | | |
| Proposed Thesis or Dissertation Title: | Web routers: An explorative performance review | |
|  | | |
| Brief Description of Research: | Performance evaluation of data structures and algorithms for web routing as used in web servers and web service frameworks. | |
|  |  | |
| Due Date for Thesis or Dissertation Submission: | |  |
|  | | |

Table of Contents

[Web routers: An explorative performance review 1](#_Toc467194256)

[Table of Contents 2](#_Toc467194257)

[Proposal Checklist 2](#_Toc467194258)

[Supervisor Team Details 4](#_Toc467194259)

[Main Supervisor Approval 5](#_Toc467194260)

[Further Approvals for this Proposal 6](#_Toc467194261)

[Web routers: An explorative performance review 7](#_Toc467194262)

[Introduction 7](#_Toc467194263)

[Literature Review 7](#_Toc467194264)

[The request-response cycle 9](#_Toc467194265)

[Current routing approaches 10](#_Toc467194266)

[Research Objectives/Questions 12](#_Toc467194267)

[Research context 12](#_Toc467194268)

[Method 12](#_Toc467194269)

[Timetable 14](#_Toc467194270)

[Health and Safety 15](#_Toc467194271)

[Budget 15](#_Toc467194272)

[References 15](#_Toc467194273)

# Proposal Checklist

*Copy (Ctrl+C) this ticked box*  *then highlight an answer box, and Paste (Ctrl+V) the ticked box in place of the answer box.* ***Sign and date using a pen.***

|  |  |
| --- | --- |
| 1. I have taken part in a Mutual Expectation Agreement (MEA) meeting with my supervisory team.   Date of meeting: |  Yes  / / |
| 1. I have given a seminar on my research proposal.   Date of seminar: |  Yes  No  / / |
| 1. I have booked (or completed) training in how to use the Endnote bibliography software. |  Yes  No |
| 1. I agree to consult with and obtain approval from the Human Ethics Committee BEFORE starting any empirical, lab, practical or field work, and I will comply with the conditions specified in the approval. |  Yes  NA |

**Student**  **Supervisor**

**Signature:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Signature:\_\_\_\_\_\_\_\_\_\_\_** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Date Proposal  
Submitted:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Date:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Supervisor Team Details

**Main Supervisor:**

|  |  |
| --- | --- |
| **Status:** | **Supervisor** |
|  | (Supervisor, Co-Supervisor) |
| Name: | Stuart Charters |
| Position Held: | Senior Lecturer |
| Email Address: | Stuart.charters@lincoln.ac.nz |

**Other Supervisors:**

|  |  |
| --- | --- |
| **Status:** | **Co-Supervisor** |
|  | (Associate Supervisor, Co-Supervisor) |
| Name: | Walt Abell |
| Position Held: | Senior Lecturer |
| Email Address: | Walter.abell@lincoln.ac.nz |
| Company & Postal Address (if External): |  |
|  |  |
| **Status:** |  |
|  | (Associate Supervisor, Associate Co-Supervisor, Advisor) |
| Name: |  |
| Position Held: |  |
| Email Address: |  |
| Company & Postal Address (if External): |  |
|  |  |
| **Status:** |  |
|  | (Associate Co-Supervisor, Advisor) |
| Name: |  |
| Position Held: |  |
| Email Address: |  |
| Company & Postal Address (if External): |  |

# Main Supervisor Approval

*I confirm the following:*

|  |  |
| --- | --- |
| 1. A full cost budget (itemised and listed by calendar year) has been prepared for this research project and is attached. I have examined the budget, and it appears to be suitable. |  Yes |
| 1. Equipment or facilities required for completion of the project are available or can be obtained (within contraints applying to purchase of capital equipment) using the project funding applied for or contracted. |  Yes  No   NA |
| 1. The outcome of this research project could produce commercialisable Intellectual Property (protectable by patent or not).   If YES, I have advised the Director of the Research & Commercialisation Office in writing (email), so that the Director can consider what further documentation may be necessary. |  Yes  No  Unsure   Yes |
| 1. I approve this proposal with the following comment: | |

Main Supervisor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Further Approvals for this Proposal

**Other Supervisors:**

|  |
| --- |
| I \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, approve this proposal with the following comment:  Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |
| I \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, approve this proposal with the following comment:  Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |
| I \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, approve this proposal with the following comment:  Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |

**Postgrad Convenor of Department/School:**

|  |
| --- |
|  |
| I \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, approve this proposal with the following comment:  Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |

**Faculty Postgrad Convenor:**

|  |
| --- |
|  |
| I \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, approve this proposal with the following comment:  Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |

# Web routers: An explorative performance review

# Introduction

The focus of research in recent times has been on the client side of web development, as defined by the Hyper Text Transfer Protocol (HTTP) specification. The main body of research has been to produce better ways to present information, and with the interaction to the user.

TODO: reference HTTP 1.x spec

Server side focus in the past has been in making dynamic content more easily created and manipulated. The lack of interest and development in this area has left certain technologies in use by the web stack with limited work done on them; an example of this would be the web server router.

TODO: reference ORM and or templates

Web routers are the core technology that allow for execution of code in response to requests by a HTTP client. A web router does not interact with the user directly, instead it is configured by descriptions (routes) of websites which hook it into the web service. Software developers implement a web service to produce dynamic content for a website. Dynamic content and static content utilize a web server router to locate the resource handling mechanism to execute and produce a response to the HTTP request.

This research is to determine the performance of web router implementations for usage as part of a web server.

# Literature Review

The World Wide Web was conceptualised in 1989 (World Wide Web Consortium, n.d.) since then there has been a large uptake in its usage by everyone across the globe to an estimate of 3.4 billion users as of October 8th of 2016 (Internet Live Stats, 2016). With every one of those users working with the standards of Uniform Remote Locator[[1]](#footnote-2), Hypertext Transfer protocol[[2]](#footnote-3) and Hypertext Markup Language[[3]](#footnote-4) in some form or another.

During the early days, many different web browsers and servers were created, the majority of these have since long died off but the legacy that is the definition of each has not. As defined by the World Wide Web Consortium (W3C) (World Wide Web Consortium, 2014):

1. Web browser  
   A program which allows the display and execution of a web page for a user. Interacts with a web server to provide any data required. This is the most common form of client.
2. Web server  
   Retrieves files or resources from the file system or some form of backend such as a web application and sends them to the client as requested.
3. Web Server API or service  
   A standalone piece of software that will dynamically create content to send to a client. It communicates in some form to the web server to serve up content to the client.

With an upsurge in internet speeds during 1990s as demonstrated by Nielsen’s Law (Nielsen, 1998), companies and developers alike experimented with dynamic web pages allowing for user interactions not possible with static web pages alone. The Common Gateway Interface[[4]](#footnote-5) was created to allow for external program to be executed as part of the web page processing by a server. From this point on existing programming languages gained new 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 (The PHP Group) which has the primary purpose of dynamic page creation on each request by the client.

## The server

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, with only slightly different purposes and entry points. The web router resides on a server and is required component for the operation of both web servers and web APIs.



Figure 1 General HTTP request + response processing activities

A web router, primary goal is to map any incoming request from a socket to a function 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 the request. This is shown in Figure 1 General HTTP request + response processing is based upon HTTP 1.x diagram.

The tendency of web developers is to focus upon coding within the requests and manipulating of the response for the client side. For the server side the focus is upon handling the routes for a given purpose. When it comes to implementation of the libraries, frameworks and end user code there is little consideration by those who use a specific implementation and along with it, its performance. This can cause problems such as the time it takes to handle a request from getting it to responding to it back the client. These existing algorithms (e.g. linked list, B-Tree) and data structures were created for the usage within a database engine. In the context of a database they have been optimized and analysed for best performance. For a web server, these algorithms and data structures may have improved performance once they have been analysed with optimizations for this particular use case.

## The request-response cycle

At the core of a web router is the process of turning an HTTP request into a call to a procedure to handle the request and return the result along with some meta information (e.g. how long to cache it for). This process has several stages:

1. Socket listening & connection
2. HTTP request received
3. Routing to function call & execution of function
4. HTTP response creation
5. Response sending

The above list is a general overview of the different sequential parts that a request goes through on the server. Commonly it is implemented as:

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

This overview does not consider some of the problems faced by all implementations. Not all information that is required to complete a request is held within memory while the program is executing. Instead it relies upon other software (e.g. a database) to hold it. The process to communicate and retrieve this extra information is expensive compared to the time it would have been if it was in memory at the time of request. The usage of external resources is commonly implemented using blocking operations. A good example of this is file reading and writing which is critical to web servers if they serve content held within the file system. Blocking operations prevent other requests from being concurrently served.

Asynchronous execution of requests is a complex topic that can affect performance between web servers quite significantly. The difference is exemplified by Nginx and Apache2 httpd. Nginx uses asynchronous event based handling, while Apache2 utilizes a thread based approach (DigitalOcean, 2015).

When a connection has been established and the handling code is executed, 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 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[[5]](#footnote-6) and Fast-CGI[[6]](#footnote-7) processing. From this the handling mechanism for the specific route is called with it.

## Current routing approaches

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 route is. The approaches that are available to implement the routing can differ quite significantly in the behaviours that they express. These different approaches each have a different set of costs and cannot be interchanged in each context with the expectation of performance related changes occurring.

There is a variety of different methods used in implementing a web router. Common ones include: tree graphs such as a Red-Black tree graph or using a single Regular Expression (regex). A single regex can simplify the code required but will result in a limited capability. With only the host name and URI path being validated against.

TODO: needs references, Nginx for RB-Tree, PHP reference for regex expressions in a router

At the core of what a web router does is to take a set of known variables and return a function to execute with the potential to modify the known variables. These set of variables that must be utilized in each searching of the underlying structure are unique when compared with existing research into data structures which focuses primarily upon a single variable. The extension to multiple variables to check and a more complex search algorithm that may need to repeat itself mean existing data structures and algorithms may be used but modified to consider that simple comparisons do not correctly relate entries to the search parameters.

Current implementations typically use with the path from the HTTP header to perform lookups. These require the least extension to existing data structures and algorithms. Regular expressions are typically used to implement them. These cover most cases; by utilizing multiple instances of the router implementation it can be used for different HTTP methods such as GET and POST without direct support within the elements of the data structure.

Some servers support a feature known as rewriting. Rewriting is the process by which requests are modified into being another; however only internally. After a ‘rewrite’ of a request takes place it must be evaluated out as if it was a new request. Most web routers do not implement this feature because of its complex nature. The rules by which it can modify the request by can include the path, domain, time stamp, client IP address and any other HTTP request field.

In non-regex approaches, more information is stored using data structures. Such as a key in a map or to wrap the reference to the handler function. This allows the routing algorithm 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 limited research into this area was discovered in the creation of this proposal.

The implementation of the storage mechanism that the web router utilizes can take many forms including a list or a tree graph. These data structures are simple in design but have many optimization opportunities such as cache locality for children in a tree graph which can improve performance by many magnitudes (Ross & Rao, 2000).

# Research Objectives/Questions

# Research context

The research proposal being presented here has the end goal of trying to make the world-wide web faster, in general better for developers and businesses in terms. The classification for better in this case is the monetary cost of serving requests to the clients. In previous work a considerable amount of research has gone into making the web faster by focusing upon the total performance of the web request processing cycle with little investigation into optimization of server side performance in terms of the web router.

The overarching goal is to determine the performance of various web router algorithms for a given set of circumstances; what is the performance of a set of web router algorithm implementations?

From this a set of sub-questions is formulated to help reach the overall research goal.

* What are the current performance metrics associated with the request/response cycle?
* What are common algorithms that web routers use and how do they relate to those used in other fields?
* What are the performance characteristics of commonly used algorithms to implement a web router given a range of routing scenarios as input?

These sub-questions will allow the performance of each algorithm to be characterised allowing analysis to determine the focus for improvement.

# Method

The research being undertaken will be utilizing a benchmark harness to execute all tests to determine a common set of metrics and this includes how long it took to perform. The harness will provide a common interface to allow for the metrics to be gathered the same way for each implementation being tested and compared against.

The input data sets as per the benchmarking comprises two different sets of information. The first is the routes to be stored into the relevant data structure (implementation of the router) and the sets of requests to be executed by the router implementation. The routes get stored pre-start of testing and should be fully optimised before the execution of the benchmarking.

The design of the routes used in the data sets include: static paths “/my/path/goes/here” with variable number of parts “/part”, a variable number of variables “/my/path/:variable” and a catch all “/my/path/\*” for all values following the previous values. These will be combined into the forms: “/path/:vars/\*”, “/path/\*” and “/path/:vars” with path and vars being variable in number. The combination and complexity will be produced algorithmically for the purposes of getting as many corner cases as possible.

Each implementation is expected to run solely within a single thread. All input sets currently within memory prior to the execution and pre-start procedures having been executed. Each input set is executed in multiples of a ten giving: 10, 100, 1000 ext. Each result is then averaged for the purpose of analysing the time the input set took to execute and return a result is compared to other input set iterations results. This occurs without breaks and does not count the pre-start procedure execution of each implementation.

The computer that will perform the benchmarking has the following specification with regards to parts being used for the analysis:

* CPU: Intel Xeon v3 E5-2630, 8 cores at 2.4ghz base frequency and 20mb cache
* Memory: DDR4 64GB at 3200mhz
* OS: Windows 10

Storage of results and input will be done within memory. Once results are fully generated and testing has concluded for an implementation test set, it will then be stored on a hard disk. This will prevent performance penalties associated with long term storage drives appearing in results.

# Timetable

|  |  |  |
| --- | --- | --- |
| **Task** | **Week of** | **Expected length of time** |
| Router implementation (1)  Flat array | December 5th | 1 week |
| Benchmark harness  Initial creation. This is expected to be a very rough implementation that does not get anywhere close to what is needed. | December 12th | 2 weeks |
| Router implementation (2)  Tree graph, no optimizations | January 2nd | 1 week |
| Router implementation (3)  Tree graph, optimizations | January 9th | 2 weeks |
| Router implementation (4)  Regex | January 16th | 1 week |
| Benchmark harness revision  Updates based upon what has been implemented with the routers. As well as include new features to make it more complete based upon what the routers are doing. | January 23rd | 2 weeks |
| Analysis upon router implementations performance | January 30th | 5 weeks |
| Writing of dissertation  Each portion of this has an expected overlapping time line of:   |  |  | | --- | --- | | Introduction | 2-3 weeks | | Literature review | 3-4 weeks | | Method | 6-8 weeks | | Analysis | 4-5 weeks | | Conclusion | 3 weeks | | Abstract | 1 week | | March 6th | 11 weeks |
| Final feedback and revisions | May 22nd | 5 weeks |
| All work must be completed by this week. | June 26th – July 2nd |  |

# Health and Safety

No known health and safety risks exist for or by this research.

# Budget

A printing budget of $100 will be needed for final copies for submission of the dissertation.

No other expenses are expected.

# References

DigitalOcean. (2015, 28 1). *Apache vs Nginx: Practical Considerations*. Retrieved from DigitalOcean: Cloud computing designed for developers: https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations

Internet Live Stats. (2016). *Number of Internet Users (2016) - Internet Live Stats*. (Internet Live Stats) Retrieved 10 8, 2016, from Internet Live Stats: http://www.internetlivestats.com/internet-users/

Nielsen, J. (1998, April 5). *Nielsen's Law of Internet Bandwidth.* Retrieved from Nielsen Norman Group: UX Training, Consulting, & Research: https://www.nngroup.com/articles/law-of-bandwidth/

Ross, K. A., & Rao, J. (2000). Making B+- trees cache concious in main memory. *ACM SIGMOD international conference on Management of data* (p. 475486). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=335449

The PHP Group. (n.d.). *History of PHP.* Retrieved 10 8, 2016, from PHP: Hypertext Preprocessor: http://php.net/manual/en/history.php.php

World Wide Web Consortium. (2014, March 14). *How does the Internet work*, 72360. Retrieved 10 8, 2016, from World Wide Web Consortium (W3C): https://www.w3.org/wiki/index.php?title=How\_does\_the\_Internet\_work&oldid=72360

World Wide Web Consortium. (n.d.). *Help and FAQ - W3C*. (W3C) Retrieved 10 8, 2016, from World Wide Web Consortium (W3C): https://www.w3.org/Help/#inventio

1. https://tools.ietf.org/html/rfc3986 [↑](#footnote-ref-2)
2. https://tools.ietf.org/html/rfc2616 [↑](#footnote-ref-3)
3. https://www.w3.org/MarkUp/draft-ietf-iiir-html-01 [↑](#footnote-ref-4)
4. https://tools.ietf.org/html/rfc3875 [↑](#footnote-ref-5)
5. <https://trac.nginx.org/nginx/browser/nginx/src/http/ngx_http_file_cache.c?rev=953512ca02c6f63b4fcbbc3e10d0d9835896bf99> [↑](#footnote-ref-6)
6. <https://trac.nginx.org/nginx/browser/nginx/src/http/modules/ngx_http_fastcgi_module.c?rev=953512ca02c6f63b4fcbbc3e10d0d9835896bf99> [↑](#footnote-ref-7)