A Resource Monitoring Scheme for Web Applications

Prithviraj Sahu

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela - 769008, India

A Resource Monitoring Scheme for Web Applications

Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Software Engineering)

by

Prithviraj Sahu

(Roll 211CS3073)

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela - 769008, India

Acknowledgment

I would like to thank all professors of Department of Computer Science and Engineering for providing the resources and the environment needed for the successful completion of the thesis. I would like to thank the admin and staff of National Institute of Technology Rourkela for extending their help and support whenever needed.

I would also like to thank my friends and peers, who have extended their help whenever needed. Their contribution has always been significant.

Finally, I would like to take this opportunity to thank my parents, who have always been a source of inspiration and motivation for me, and also for the love they have provided in stressful periods, which has been a guiding force for the completion of the thesis.

Prithviraj Sahu

Abstract

A web application is an application that uses World Wide Web's infrastructure to deliver its service and a web browser as a client. It is accessed by users over a network such as the Internet or an intranet. Web applications are popular due to the ubiquity of web browsers and the convenience of using a web browser as a client. They have become much more complex due to the inclusion of various other scripting technologies like JavaScript, Ajax and Cascading Style Sheets etc. HTML not only describes structural semantics of a web page through markup tags, but also enables the inclusion of external resources into web documents, such as images, scripts, style sheets, media files and other objects as parts of the web page. These are called Web resources. An efficient resource monitoring method is necessary for the development of web application, because the monitoring data helps in failure detection, load balancing, scheduling strategies and performance optimization. This is in response to the growing complexity and different development practices adopted in web application development, which makes it difficult to maintain resources efficiently. In this work, a resource monitoring scheme has been proposed and implemented to monitor the web resources such as images, scripts and style sheets. The results are produced in an interactive graph based visualization. The graph shows the size, load time and frequency of access of these resources, and dependency among the links accessing a particular resource. This representation helps in decision-making process from the organization point of view. For small to medium scale web applications, the scheme can also help in some speedup.

***Keywords*:** Web application, Web resource, Resource monitoring, HTML parsing, DOM tree, Web page Rendering, Regular Expression.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The World Wide Web (WWW), simply Web, was introduced to access information from any source in a consistent and simple way. Web was viewed as a vast repository of information that provided access to a large number of users. This view of the Web was static and it has changed considerably over time. Currently, Web is a powerful platform offering a large number of tools, components and services to application developers [1].

A web application is an application that uses Web's infrastructure to deliver its service and a web browser as a client. It is accessed by users over a network such as the Internet or an intranet. Web applications are popular due to the ubiquity of web browsers and the convenience of using a web browser as a client. The key reason for the popularity of web applications is the ability to update and maintain them without distributing and installing software on potentially thousands of client computers. The inherent support for cross-platform compatibility also attracts more users to use web applications. With a single Web browser, one can view news, play online music or video, check e-mail, shop in online stores, pay bills in online banks etc [2]. What is seen on a single web browser window, is called a web page. A web page contains the information to be displayed to the user by a web application or a web site. Though these terms are referred interchangeably, web application is not the same as website. Website is a mere collection of linked web pages served from a single web domain. Like website, a web application is also a collection of web pages hosted on a server; but web application is a functional piece of software, driven by underlying business logic to display dynamic and interactive contents. Web application takes requests from users through the client side web browser, processes the requests at the server side and returns the results to the client side browser to be displayed to the user.

The Web uses web page to display contents to the users through a web browser, and to navigate to other web pages via hyperlinks. Hypertext Markup Language (HTML) is the main markup language used for creating web pages and other information that can be displayed in a web browser. Web browsers coordinate web resources centered around the written web page, such as style sheets, scripts and images, to present the web page. Apart from describing structural semantics of a webpage through markup tags, HTML also enables the inclusion of external resources into web documents, such as images, videos and other objects as parts of the webpage. The general structure of web pages and their content are defined in HTML, while its final presentation and style are in the domain of CSS (Cascading Style Sheets) [2]. So a Web page contains not only text, but may also contain scripts, images, flashes, style sheets, media files, and other documents.

Web application monitoring is the process of testing and verifying that end-users can interact with a web application or website. Web application monitoring is often used by organizations to ensure that their sites are live and responding [3]. If a web application performs poorly and organization is not even aware of it, then it will be very hard to realize the goals. This is where an efficient web application performance monitoring solution can help. The management and collaboration of resources is also dependent on the monitoring data that manipulates every node status, historical CPU and memory latency, network loading between sites [4].

Web resource is a resource identified by a Uniform Resource Identifier (URI), residing on the Internet that can be accessed through Hypertext Transfer Protocol(HTTP) as part of the protocol stack, either directly or via an intermediary. An efficient resource monitoring method is necessary for the development of a web application, because the monitoring data helps in failure detection, load balancing, scheduling strategies, and performance optimization. Along with monitoring data that manipulates for node status, CPU and memory latency, and network load time [4]; monitoring web resources should also be considered for their utilization, size, load time, and dependency [5,6].

It is important not only to get results from monitoring or performing tests, but also to represent those results in a form that can be easily understood. There are a lot of tools and utilities that are present to monitor and test the performance of web applications. But the result produced is often not easy to understand and difficult for decision-making. The benefits of all the information produced in the result may not be properly realized because managers and customers may be increasingly overloaded with information in electronic form. Much of the information got from these results are mainly symbolic in nature consisting of numbers and texts. Processing this kind of information needs a lot of effort because it involves rule-based reasoning in which data are abstracted into values, that in turn, are given meaning through formal rules and deliberative analysis [7].

Humans have evolved great visual and spatial skills including the abilities to detect edges and discontinuities, things that stand out, variations in color and shape and motion; to recognize patterns; and to retrieve information using visual cues. This suggests that a solution to information overload could be to present information in ways that engage the use of associative system such as visual recognition. By drawing on highly developed skills of perceptual sense making of humans, the old adage that "a picture worth a thousand words" may be replaced with "a picture worth a thousand rows of data" [7]. The visual perspective and information context influence decision processes and outcomes by changing the decision-making frame−that is, what information a decision maker uses and how he or she uses it to gain insights and take decisions.

1.2 Motivation

Web applications often include and refer to a lot of resources such as images, scripts, and other files stored on the web. These resources contribute to the user interface and add dynamic behaviour to the web application. Due to the popularity of the web based applications, today there exist extensive collection of scripting libraries to meet the various needs of the application. Hence it has become very important to consider the resource usage into account when testing a web application. Consider a script used to provide some features of the application. Features may vary in terms of use and importance. Some of the features are essential while others are not that of primary interest and the application can still function without them. However if a change is made to the script then all the features using the script will be affected. As the development of the different parts of a web application can be done independent of each other and a lot of resources are involved, hence it is not uncommon to find unexpected results due to improper use of resources. In case of many testing utilities, the result produced is often not easy to understand and difficult for decision-making. To address these problems, an interactive graph based resource monitoring scheme is presented.

1.3 Objectives

The objectives of the work are

- to find all the Web resources referred in the Web application.
- to find the number of times each resource is referred, with its size, load time.
- to control the application remotely, as an independent application.
- to represent the resources and the dependency among them in a graph based visualization structure.
- to recommend caching of resource based on its dependency, size and number of times it is referred.

1.4 Organization of the Thesis

In this chapter, a brief introduction to the problem domain, motivation for a resource monitoring scheme and objectives of the work are discussed. Organization of the remaining part of the thesis and a brief outline of the chapters in the thesis are given as follows.

Chapter 2 discusses about some existing web resource monitoring schemes and tools. Chapter 3 discusses about the proposed resource monitoring scheme. This chapter discusses about the different modules and algorithms that have been proposed for resource monitoring of web applications. Chapter 4 discusses about the different technologies used for implementation of the proposed resource monitoring scheme and results produced from the implementation. Finally, chapter 5 summarizes the main contributions of this thesis and provides a view on future directions for this work.

Chapter 2

Web Applications, Web Resources and Monitoring Schemes

2.1 Introduction

Web applications are becoming more and more popular, and at the same time they are becoming complex through the inclusion various technologies in their development for satisfying the user needs. Now every organization is trying to make its application rich. To achieve this richness, many new development technologies are adapted. These new technologies require the inclusion of various other resources, which are referred here as web resources. So it becomes very crucial to monitor these web resources and their impact on the performance of the web application.

2.2 Web Applications

A Web application is an application that uses a web browser as a client and is accessed by users over a network such as the Internet or an intranet. It can be of a 2-tier client server architecture, or a 3-tier architecture consisting of a User Services(Presentation) tier, a Business Services tier, and a Data Services tier. The main distinction between 3-tier architecture and 2-tier client-server architecture is that, with a 3-tier architecture, the business logic is separated from the user interface and the data source. Breaking up applications into these separate tiers or sections can reduce the complexity of the overall application, and results in applications that can meet the growing needs of today's businesses. N-tier applications are just 3-tier applications that might further sub-divide the standard User Services, Business Services, or Data Services tiers.

2.2.1 Rich Web Applications

Rich Web Applications, also called Rich Internet Applications (RIA), are applications featuring responsive user interfaces and interactive capabilities which can make programs easier to use and more functional, thus enhancing user experience [8]. They provide benefits that are more usually associated with desktop applications: availability, interactivity, responsiveness, utility, and richness [9,10]. Today's Web applications tend to become rich through the use of CSS, JavaScript, Ajax(Asynchronous JavaScript and XML), and jQuery. CSS gives Web site developers more control over how browsers display pages.CSS is used to create style sheets that define how different page elements, such as headers and links, appear. JavaScript interacts with HTML code and makes Web pages and Ajax applications more active. For example, the technology can cause a linked page to appear automatically in a popup window or let a mouse rollover change text or images. Ajax is a new way to think, design, develop, and a new style to program Web applications. Ajax is a new technique that uses a set of open standards technologies, with support by cross-platform browser and cross-platform compatibility. Because JavaScript is a cross-platform scripting language, Ajax applications require no plug-ins [8, 11].

2.2.2 Web Application Development Technologies

The typical structure of a Web application consists of HTML data displayed to the user, client-side scripts that run on the client and may interact with the user, and server-side scripts that perform processing on the server and typically interact with databases [1]. Client Side Scripting is the type of code that is executed or interpreted by browsers. Client side Scripting is generally viewable by any visitor to a site. Common client side scripting technologies used are HTML, CSS, JavaScript, Ajax, and jQuery. Server side Scripting is the type of code that is executed or interpreted by the web server. Server Side Scripting is not viewable or accessible by any visitor or general public. Common server side scripting technologies used are ASP, ASP.NET, Cold Fusion, JSP, PHP, Ruby, Perl, Python *etc.* [12].

2.2.3 How Browsers load Web pages?

To load a page, a browser typically first downloads a main HTML object that defines the structure of the page. Next, it may download a CSS object that describes the presentation of the page. The main HTML object may embed many Javascript objects that are executed locally to interact with a user. As the page is being rendered, an HTML or a Javascript object may request additional objects, such as images and Javascripts. This process continues until all relevant objects are loaded [13]. Figure 2.1 [14] shows a diagramatic representation of the processes involved in loading a wb page. The detail process of how browsers load web pages is described next [14].

HTML Parser: The HTML Parser transforms the raw HTML page to a document

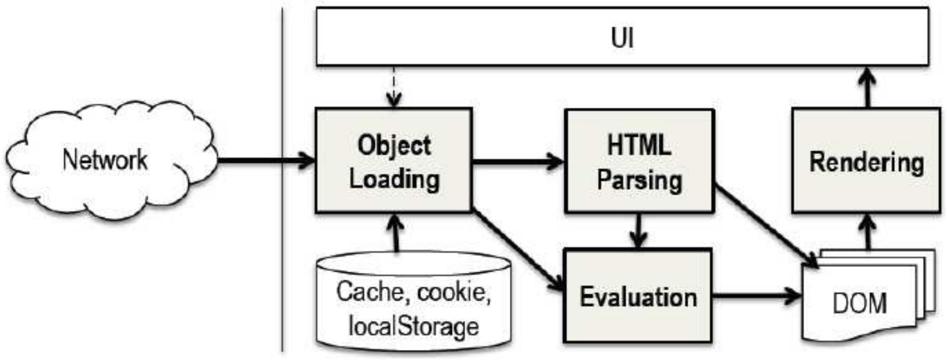


Figure 2.1: Process of Web page loading

object model (DOM) tree. A DOM tree is an intermediate representation of a web page. Figure 2.2 shows a DOM tree representation of an HTML document. The nodes in the DOM tree represent HTML tags, and each node is associated with a set of attributes. The DOM tree representation provides a common interface for programs to manipulate the page.

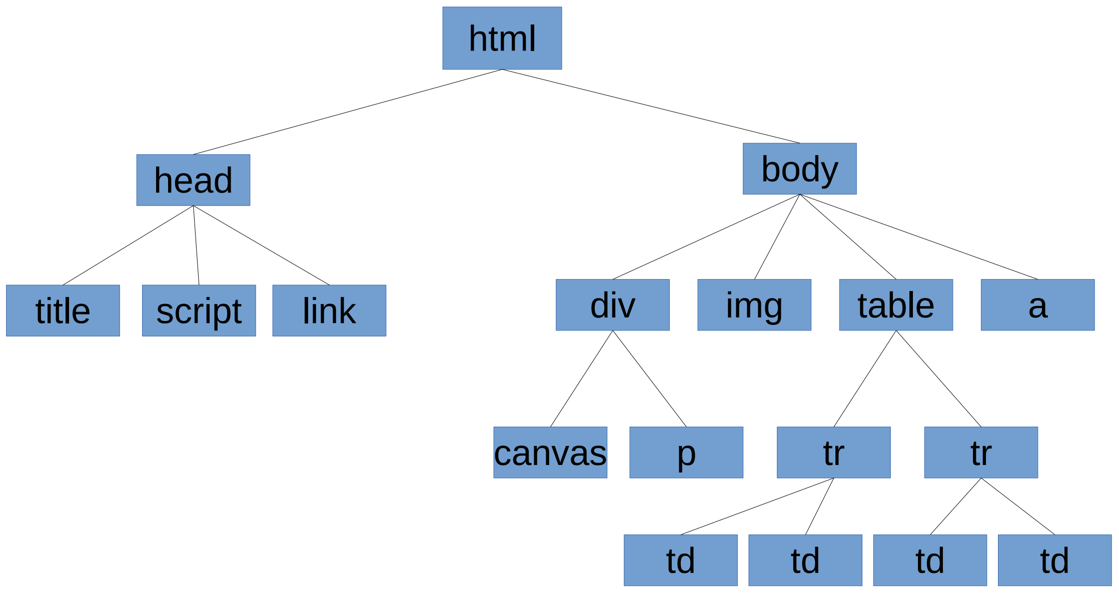


Figure 2.2: DOM tree representation of HTML document

Object Loader: The Object Loader fetches objects requested by the user or those embedded in the HTML page. The objects are fetched over the Internet using HTTP, unless the objects are already present in the browser cache. The embedded objects fall under different mime types: HTML, external JavaScript, external CSS, Image, and Media. Inline JavaScript and inline CSS do not need to be loaded.
Evaluator: Two of the five embedded object types, namely, JavaScript and CSS, require additional evaluation after being fetched. JavaScript adds dynamic content to web pages. Evaluating JavaScript involves manipulating the DOM, e.g., adding new nodes, modifying existing nodes, or changing nodes styles. CSS are used for specifying the

presentational attributes (e.g., colors and fonts) of the HTML content. Evaluating a CSS rule involves changing styles of DOM nodes.
Rendering Engine: Browsers render web pages progressively as the HTML Parser builds the DOM tree. Rendering involves two processes: layout and painting. Layout converts the DOM tree to the layout tree that encodes the size and position of each visible DOM node. Painting converts this layout tree to pixels on the screen.

2.3 Web Resources

Web resource is a resource identified by a Uniform Resource Identifier (URI), residing on the Internet that can be accessed through Hypertext Transfer Protocol(HTTP) as part of the protocol stack, either directly or via an intermediary [5, 6]. Web resources generally include images, style sheets, scripts, links, objects, documents, media files *etc.* They affect the way the web pages are displayed in the browser, their load time, and their functionalities. The main web resources that affect the display and behaviour of a web application are scripts, style sheets, and images. So in the proposed approach we have considered only these resources into account.

2.3.1 Web Resource Monitoring

Web page load time is a key performance metric that many techniques aim to reduce. Unfortunately, the complexity of modern web pages makes it difficult to identify performance bottlenecks. So an efficient resource monitoring method is necessary for the development of web application, because the monitoring data helps the failure detection, the load balancing, the scheduling strategies, and the performance optimization [5, 6]. Web resource monitoring comes under web application monitoring, which is the process of testing and verifying that end-users can interact with a web application or website.

Web application monitoring is often used by business to ensure that their sites are live and responding. If your web application performs poorly and you are not even aware of it, it will be really hard to realize these goals. This is where an efficient Web resource monitoring solution can help [3]. An efficient web resource monitoring scheme include the details about the types of files, file sizes, load times, total website speed and other details about every single element of a web page (HTML, JavaScript and CSS files, images, *etc.*).

2.3.2 Web Resource Performance Factors

As explained in section 2.2.3, the root HTML document may refer to other resources, such as images, scripts or style sheets. Each of these resources must be fetched with a subsequent HTTP request. Each HTTP request creates network traffic between the client and server, which adds to performance overhead. Reducing number of referenced resources will consequently reduces the number of HTTP requests, which will improve application performance. Ajax allows for HTTP requests to be dispatched asynchronously with JavaScript code, without reloading the entire HTML page along with all of its referenced resources. Hence Ajax reduces the total number of HTTP requests and greatly improves frontend performance. Web browsers also cache resources to reduce the number of HTTP requests. This means that a web browser is able to store certain resources (such as images, style sheets and scripts) locally, instead of fetching them over the network each time. This behavior is controlled by a number of requests and response headers. In response to a conditional GET, if the resource has not changed, the application server may return a 304 Not Modified response with no body; which reduces the amount of transmitted data. An application server may choose to supply the Expires and/or Cache-Control response headers with responses, which are used to signal the client that a resource should only be re-retrieved after a particular period of time has passed. Thus caching should be used, whenever possible, to improve application performance. Style sheets should be cached by the browser to reduce the total number of HTTP requests. Caching of style sheets is enabled by using external style sheets, which allow for the style sheet to be requested separately from the main document, and by appending the appropriate caching headers to responses. Because browsers often utilize progressive rendering, i.e. render whatever content is available as soon as possible, misplaced references to style sheets can delay the rendering of a web page by forcing the browser to defer rendering of the entire document until those references have been resolved. It is thus appropriate to put references to style sheets at the top of the HTML document to allow for proper progressive rendering and, consequently, improved application performance. External style sheets should not be included to a single HTML document more than once. Duplicate style sheets delay the progressive rendering of the web page. Like style sheets, JavaScript scripts should be externalized and cached whenever possible to improve performance. But unlike style sheets, scripts should be placed at the bottom of an HTML document for better performance. This is because script execution not only blocks parallel downloads of resources, but also effectively disables progressive rendering of elements appearing after the script. Furthermore JavaScript is a rich programming language that allows the developer to use arbitrary names for variables and functions, add comments, and format code with an arbitrary amount of whitespace (spaces and tabs). Therefore script files can become very large. This will result in reduced performance. Finally, one should make sure that an external JavaScript, like an external style sheet, is never included to a single HTML document more than once. Duplicate scripts require both duplicated HTTP requests and processing effort, which implies reduced application performance [15, 16].

2.3.3 Web Resource Extraction from Web pages

Web resources are included in the HTML code as part of their tags. The tags that contain the resources generally fall under the following categories: scripts in <script> tag and 'src' key, images in <img> tag and 'src' key, style sheets in <link> tag and 'href' key, and hyperlinks in 'href' key of <a> tag or 'action' key of <form> tag. The resources can be easily retrieved from these tags using the keys and by matching their file extensions. Scripts have ".js" file extension, style sheets have ".css" file extension, and images have many file extensions such as ".jpg", ".png", ".gif", ".bmp", ".ico" *etc.*

As HTML is not strict enough, it increases the complexity of resource extraction from the tags. For example HTML can descript the links of urls and images like following:
<img src="nitlogo.gif" width=10 height=10 ...>
<IMg width=10 height=10 src='../nitlogo.gif'...>
<A href="http://nitrkl.co.in">
<a class="default" href='http://nitrkl.co.in'>
Although server can compile and run it successfully, but it is difficult to extract. If we extract resources by using strict word matching, it will be very difficult. Because there are so many character combinations as img, Img, iMg, imG and so on. Rather, a simple solution will be to use regular expressions. So regular expressions can be used in place of possible tag names which can be efficiently used to extract desired content from tags.

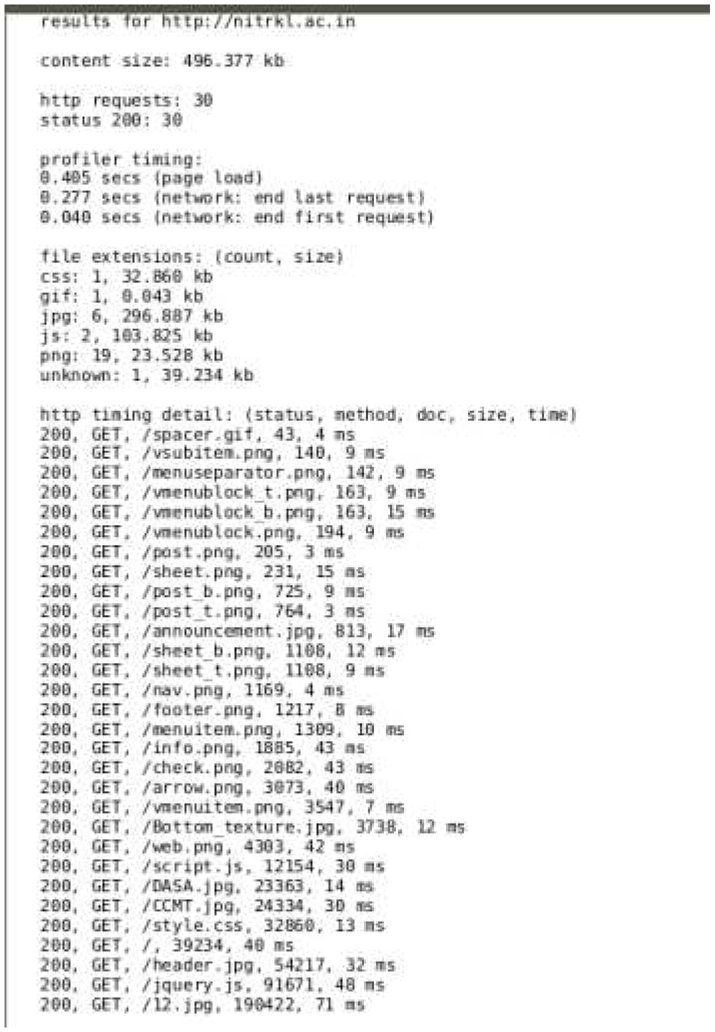
In order to get the image links which form <img src="..."">, the regular expression could be designed like <[Ii][Mm][Gg]([ˆ>]*)\\s*>; as the 'src' property is not only applied in <img> tags but also in somewhere else like <script>. It can filter the content from <img...> tag. According to the same principle regular expressions can also be designed like [Ss][Rr][Cc]\\s*=\\s*[\' "([ˆ\' "]*)[\' "]. Finally the content in src="..." or src='... ' could be extracted [17].

2.4 Monitoring Schemes and Related Work

Corey Goldberg [18] built Selenium-profiler, a web/http profiler, with Selenium-RC and Python. It profiles page load time and network traffic for a web page. The profiler uses Selenium-RC to automate site navigation (via browser), proxy traffic, and sniff the proxy for network traffic stats as requests pass through during a page load. It is useful to find out the number of requests a page makes, status codes returned, page load time, number of each file type requested and their size. Figure 2.3 shows sample output of running selenium profiler of “http://nitrkl.ac.in”.

Pingdom [19] is another Full Page Test tool used to perform a website speed test to analyze the speed of websites and suggests how to make them faster. It lets the user identify what about a web page is fast, slow, too big. It helps web developers to optimize the performance of their websites and web applications. It shows file sizes, load times, total website speed and other details about every single element of a web page (HTML, JavaScript and CSS files, images, *etc.*).These data can be used to identify performance bottlenecks and necessary steps can be taken to remove these bottlenecks.

Xiao Sophia Wang *et al.* [14] presented WProf, a lightweight in-browser profiler that produces a detailed dependency graph of the activities that make up a page load. WProf is based on a model developed to capture the constraints between network load, page parsing, JavaScript/CSS evaluation, and rendering activity in popular browsers. The key role of WProf profiler is to record timing and dependency information for a page load. While the dependency information represents the structure of a page, the timing information captures how dependencies are exhibited for a specific page load; both are crucial to pinpoint the bottleneck path.



```
results for http://nitrkl.ac.in

content size: 496.377 kb

http requests: 30
status 200: 30

profiler timing:
0.405 secs (page load)
0.277 secs (network: end last request)
0.040 secs (network: end first request)

file extensions: (count, size)
css: 1, 32.860 kb
gif: 1, 0.043 kb
jpg: 6, 296.887 kb
js: 2, 103.825 kb
png: 19, 23.528 kb
unknown: 1, 39.234 kb

http timing detail: (status, method, doc, size, time)
200, GET, /spacer.gif, 43, 4 ms
200, GET, /vsubitem.png, 140, 9 ms
200, GET, /menuseparator.png, 142, 9 ms
200, GET, /vmenublock_t.png, 163, 9 ms
200, GET, /vmenublock_b.png, 163, 15 ms
200, GET, /vmenublock.png, 194, 9 ms
200, GET, /post.png, 205, 3 ms
200, GET, /sheet.png, 231, 15 ms
200, GET, /post_b.png, 725, 9 ms
200, GET, /post_t.png, 764, 3 ms
200, GET, /announcement.jpg, 813, 17 ms
200, GET, /sheet_b.png, 1108, 12 ms
200, GET, /sheet_t.png, 1108, 9 ms
200, GET, /nav.png, 1169, 4 ms
200, GET, /footer.png, 1217, 8 ms
200, GET, /menuitem.png, 1309, 10 ms
200, GET, /info.png, 1885, 43 ms
200, GET, /check.png, 2082, 43 ms
200, GET, /arrow.png, 3073, 40 ms
200, GET, /vmenuitem.png, 3547, 7 ms
200, GET, /Bottom_texture.jpg, 3738, 12 ms
200, GET, /web.png, 4303, 42 ms
200, GET, /script.js, 12154, 30 ms
200, GET, /DASA.jpg, 23363, 14 ms
200, GET, /CCMT.jpg, 24334, 30 ms
200, GET, /style.css, 32860, 13 ms
200, GET, /, 39234, 40 ms
200, GET, /header.jpg, 54217, 32 ms
200, GET, /jquery.js, 91671, 48 ms
200, GET, /12.jpg, 190422, 71 ms
```

Figure 2.3: Result of Selenium Profiler for http://nitrkl.ac.in

Zhichun Li *et al.* [13] developed WebProphet, a system that automates performance prediction for web services. They first extracted the dependency between web objects. Modern web pages contain many types of objects, including HTML, JavaScript, CSS, and image. These embedded objects are downloaded via separate requests. One object depends on the other if the former cannot be downloaded until the latter is available. WebProphet implements a model based on extracted dependency to simulate the page load process of a web browser, which enables accurate page load time prediction under changes to any web objects.

Zhang Xu *et al.* [17] presented a model to extract web page information based on tags. They used regular expressions to match tags, and from the tags, they extracted the source. URL judgments were used to find out links.

2.5 Conclusion

In this chapter, recent trends in web application development are discussed. Web resources, their monitoring, performance factors and extraction policies are presented. Various web resource monitoring schemes are also discussed in this chapter.

Chapter 3

Web Resource Monitor Scheme

3.1 Introduction

The proposed framework is for monitoring the web resources such as scripts, style sheets and images. It determines the size, load time, frequency of its access per link, dependency among links for a particular resource and redundancy, if any, in accessing a resource.

3.2 Proposed Scheme

As part of this work, a testing framework to monitor the performance of web application from the users perspective is presented. The proposed monitoring framework provides performance of the web application taking the resources it uses into consideration. In general, all web applications undergo a series of change in states. Each state is monitored using this test framework and the performance parameters such as page load time, response time, usage frequency for each state is recorded. Dependency among the states are also noted. Based on this information, the monitoring framework presents the result in form of a graph based visualization.

The framework has been designed in a manner so that it can operate as an independent web application. It can also be used at the organization's server along with the web application under test. The usage scenario is depicted in Figure 3.1.

3.2.1 Framework Modules

The framework consists of five modules as shown in Figure 3.2. They are described as follows:

A. ***Client Interface Module***

Client Interface Module serves as the entry-point to the monitoring framework. It provides the primary interface for user interactions with the Web application to be

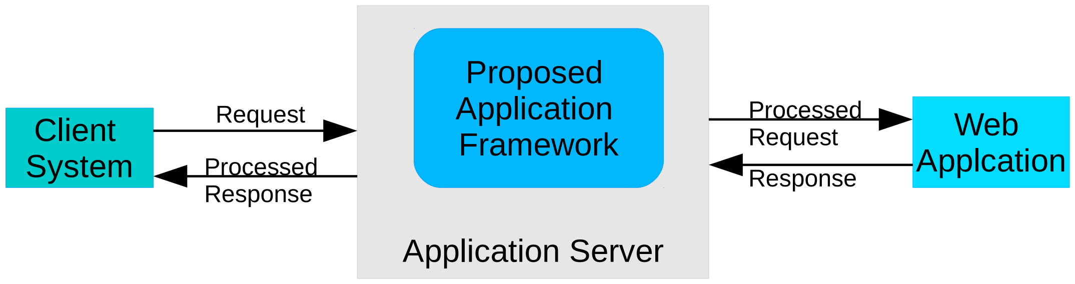


Figure 3.1: Usage Scenario

monitored. At the beginning a test user specifies a link of the Web application to be monitored. Then it takes the request and forwards it to the Request Processing Module.

B. ***Request Processing Module***

Request Processing Module is responsible for handling the requests made by the test users. It processes the different attributes of the request such as timestamp, and end point of the request. This information is recorded in the data store and the processed request is sent to the Application Interaction Module.

C. ***Application Interaction Module***

Application Interaction Module accepts the processed request from the Request Processing Module and submits this request to the Web application. It also accepts the response from the Web application. This module records the page response time and load time of various resources included in the page and this information is stored in the data store for further processing. Due to the easy scalability of the web platform, Web applications can become too complex. Hence the change in the state and request or access of a resource throughout the monitoring process is maintained in a graph structure.

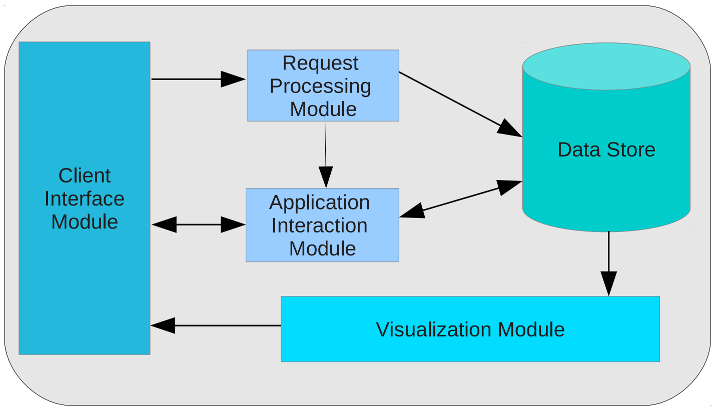


Figure 3.2: Components of Proposed Framework

D. ***Data Store Module***

Data Store Module acts as the backend for the framework. It stores the data generated from the other modules and provides them when requested. For every request made, it stores the timing information and the location of its endpoint. It also records the different attributes associated with the response.

E. ***Visualization Module***

In case of many testing and monitoring utilities, the result produced is often not easy to understand and difficult for decision-making. Hence in the framework a dedicated Visualization Module is included to provide the information in a productive way. The desired information is represented through an interactive graph structure.

3.2.2 Algorithms

When the user starts interacting with the framework, he will have to submit the request link. This link is stored in $RequestLink$. Algorithm 3.1 then uses this $RequestLink$ and returns the link details. The $parseHTML()$ function returns the HTML content fo the $RequestLink$.

Algorithm 3.1 *getLinkDetails*(*RequestLink*)

Input: *RequestLink*

Output: *LinkDetails*

```
1: Start time of request: $t_{start} \leftarrow currentTime()$
2: $Html \leftarrow pasreHTML(RequestLink)$
3: End time of response: $t_{end} \leftarrow currentTime()$
4: $RequestTime \leftarrow t_{start}$
5: $ResponseTime \leftarrow t_{end} - t_{start}$
6: $LinkDetails = (Html, RequestTime, ResponseTime)$
7: return $LinkDetails$
```

Algorithm 3.2 *getResources*(*Html*)

Input: *Html*

Output: *ResourceList*

```
1: $TagList \leftarrow \{img_{re}, a_{re}, script_{re}, link_{re}, style_{re}\}$
2: $KeyList \leftarrow \{src_{re}, href_{re}\}$
3: for each $Tag \in TagList$ do
4:     for each $Key \in KeyList$ do
5:         $ResourceList \leftarrow ResourceList \cup Value(Key(Tag))$
6:     end for
7: end for
8: return $ResourceList$
```

Algorithm 3.2 uses *Html* document produced in Algorithm 3.1 to find out the resources from the tags. As HTML is not a strict language, regular expressions have been used to get the resources from the tags. The framework mainly focuses on images, external scripts and external style sheets. So inline scripts and inline style sheets are not considered into account.

Algorithm 3.3 *getResourceGroups*(*Html*)

```
Input: $Html$
Output: $ResourceGroups$
1: for each $ResourceLink \in ResourceList$ do
2:     if Extension of $ResourceLink = Ext_{img}$ then
3:         $Image \leftarrow Image \cup ResourceLink$
4:     else if Extension of $ResourceLink = Ext_{script}$ then
5:         $Script \leftarrow Script \cup ResourceLink$
6:     else if Extension of $ResourceLink = Ext_{style}$ then
7:         $Style \leftarrow Style \cup ResourceLink$
8:     end if
9:     $HashFile \leftarrow HashFunction(ResourceContent)$
10:    $Hashes \leftarrow Hashes \cup HashFile$
11: end for
12: $ResourceGroups \leftarrow \{Image \cup Script \cup Style \cup Hashes\}$
13: return $ResourceGroups$
```

Algorithm 3.3 generates groups of resources, such as images, scripts, styles, and hashes. Groups are created by considering the file extensions of the resources. Hashes are created and stored to check two file contents are same. It returns the resource groups associated with the *RequestLink*.

Algorithm 3.4 *createResourceGraph*(*LinkList*)

```
Input: $LinkList$
Output: $ResourceGraph$
1: Create Empty $ResourceGraph$
2: $Node_{prev} \leftarrow$ Null
3: for each $Link \in LinkList$ do
4:     $Node_{Link} \leftarrow createNode(Link)$
5:     $NodeType \leftarrow$ Link
6:     if $Node_{prev}$ != Null then
7:         $addEdge(Node_{prev}, Node_{Link})$
8:         $EdgeType \leftarrow$ LinkToLink
9:     end if
10:    $Html \leftarrow getLinkDetails(Link)$
11:    $ResourceList \leftarrow getResources(Html)$
12:    for each $Resource \in ResourceList$ do
13:        $Node_{Resource} \leftarrow createNode(Resource)$
14:        $NodeType \leftarrow$ Resource
15:        $Node_{Group} \leftarrow createNode(resourceGroup(Resource))$
16:        $NodeType \leftarrow$ ResourceGroup
17:        $addEdge(Node_{Link}, Node_{Group})$
18:        $EdgeType \leftarrow$ LinkToGroup
19:        $addEdge(Node_{Group}, Node_{Resource})$
20:        $EdgeType \leftarrow$ GroupToResource
21:    end for
22: end for
23: return $ResourceGraph$
```

Algorithm 3.4 generates the resource graph from the information collected in previous algorithms. *ResourceGraph* signifies the area to represent nodes and edges, and the mapping among them. Method *createNode*() creates a new node (updates if already present). Method *addEdge*() creates a new edge between two nodes. At the *ResourceGraph* contains all resource nodes mapped to their respective links and resource groups.

The visualization module requires that the *ResourceGraph* should be interactive and responsive to the user. Algorithm 3.5 provides the details of the process of highlighting the link nodes when ever mouse pointer is hovered on a resource node.

Algorithm 3.5 *graphInteraction*(*ResourceGraph*)

Input: *ResourceGraph*

```
1: if Mouse Hover on $Node$ then
2:     if $NodeType$ = Link OR ResourceGroup then
3:         $expand(Node)$
4:     else if $NodeType$ = Resource then
5:         for each $Link \in LinkList$ do
6:             if $Resource \in getResources(Link)$ then
7:                 Highlight node($Link$)
8:             end if
9:         end for
10:    end if
11:    if Mouse Clicked outside $Node$ then
12:        $shrink(ResourceGraph)$
13:    end if
14: end if
```

Each resource referred by the application throughout the monitoring phase, are maintained in a graph structure along with the states of the application. To assist in resource monitoring,a method to get the list of states that would be affected due to modifications to a given resource is provided. As in many application states are interdependent, hence a depth parameter is included to control the search of affected states in the neighbouring states. Algorithm 3.6 provides the details of the process.

Algorithm 3.6 $getDependencyList(ResourceGraph, State, Depth)$

```
Input: $ResourceGraph, State, Depth$
Output: $DList$
1: $DList \leftarrow DList \cup State$
2: if $Depth = 0$ then
3:     return $DList$
4: end if
5: $AdjList \leftarrow \{S|\ (State, S) \in ResourceGraph\}$
6: for each $S \in AdjList$ do
7:     $DList \leftarrow DList \cup getDependencyList(ResourceGraph, S, Depth - 1)$
8: end for
9: return $DList$
```

3.3 Conclusion

In this chapter, the proposed framework is discussed in detail. Here five modules of the framework are discussed: Client Interface Module, Request Processing module, Application Interaction module, Data Store Module, and Visualization Module. An algorithm to determine the dependency list for a particular resource from the graph produced in visualization module is also presented in this chapter.

Chapter 4

Web Resource Monitor

4.1 Introduction

A working implementation of the monitoring scheme described in the previous chapter has been developed and is being referred as “Web Resource Monitor”. This chapter describes the important aspects associated with the implementation. The implementation uses a number of both client and server-side libraries to get the results. The output produced by the Web Resource Monitor is provided and explained in details in section 4.3.

4.2 Implementation Details

The proposed monitoring scheme is used as a server program. For developing the monitoring server the web.py [20] python web framework has been used. The Request Processing Module, the Application Interaction Module, and the Visualization Module described in the previous chapter are part of the monitoring server only. The Client Interaction Module is also provided by the monitoring server. This is the interface through which a user or developer is able to provide the URL of the web page or application to be monitored using the proposed monitoring system.

The data store module is materialized using the MongoDB NoSQL database. MongoDB is a document database that provides high performance, high availability, and easy scalability. A MongoDB deployment hosts a number of databases. A database holds a set of collections. A collection holds a set of documents. A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collections documents may hold different types of data [21]. The reasons for choosing a NoSQL database over traditional RDBMS is described next.

Relational databases are well established methods that allow applications to store data through a standard data modeling and query language (Structured Query Language). When the concept began, storage was expensive and data schemas were fairly simple to express. Since the rise of the web, the volume of data stored about users, objects, products and events has exploded. Data is also accessed more frequently, and is processed more intensively for example, social networks create hundreds of millions of customized, real-time activity feeds for users based on their connections' activities.

Even rendering a single web page or answering a single API request may take tens or hundreds of database requests as applications process increasingly complex information. Interactivity, large user networks, and more complex applications are all driving this trend.

In response to this demand, computing infrastructure and deployment strategies have also changed dramatically. Low-cost, commodity cloud hardware has emerged to replace vertical scaling on highly complex and expensive single-server deployments. And engineers now use agile development methods, which aim for continuous deployment and short development cycles, to allow for quick response to user demand for features. Relational databases were never designed to cope with the scale and agility challenges that face modern applications and aren't built to take advantage of cheap storage and processing power that's available today through the cloud [22]. Hence NoSQL databases with the features such as dynamic schemas, Auto-sharding, replication and integrated caching offers good advantages over traditional RDBMS for large-scale web applications. As the monitoring scheme keeps track of the various resources included or references in a web page and the content it stores is frequently accessed and processed hence a NoSQL database serves as a better option.

In general, the data store module keeps track of the requests and responses. Two collections are maintained in a MongoDB database called TestServerDB to maintain the information. One Collection named *links*, keeps track of the requested resource on the web. In this case, it could be the link to the web application or any resource the application uses. This information is obtained from the Request Processing Module. The *links* collection also records the time at which the request was received and the time taken in getting a response *i.e.* the response time. Another collection named *resources*, keeps track of the resources involved with the requested web page. At present it only monitors the JavaScript files, CSS style sheets, some image files and the links associated with the requested web page. All this information is recorded on per request basis. This information is used later by the visualization module to generate a report.

The Request Processing Module is a simple component that accepts an URL to the requested web page from the user through the client interface via a GET request and puts this information into the TestServerDB database. It then provides this information to the Application Interaction Module.

Now the Application Interaction Module sends the request on behalf of the user and waits for the response. Then the response obtained is processed and provided to the user through the client interface. The processing involves the following steps. First the response time is determined using the time difference in making the request and getting the response and the information is inserted into the *links* collection in the TestServerDB database. Next if the response is in form of a HTML document then it is parsed using Beautiful Soup [23] python library. Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with the parser to provide idiomatic ways of navigating, searching and modifying the parse tree. The entire HTML document received as part of the response is scanned for images, scripts and style sheets using their respective tags. Now for each type of resource the link to the resource, the time it takes to load the resource and the size of the downloaded file is recorded in form of a tuple and is inserted to the *resources* collection along with the link to the page that contains the resources which is the URL given by the user. The information for each resource type is kept in separate fields in resources collection. Now some URL translations are performed to make the HTML document acceptable for use by the user. Finally the processed HTML document is presented to the user through the Client Interaction Module.

In order to view an analysis of the part of the web application used, a report is prepared and presented by the visualization module. This provides the user or developer a details on the usage of resources in an easy to interpret manner. The web pages visited are considered on the basis of their request access time, and using this information a graph visualization is generated. For creating the graph, the arbor.js graph visualization JavaScript library is used. Arbor is a graph visualization library built with web workers and jQuery. Rather than trying to be an all-encompassing framework, arbor provides an efficient, force-directed layout algorithm plus abstractions for graph organization and screen refresh handling. It leaves the actual screen-drawing to the developer so that the developer can decide on using canvas, SVG, or even positioned HTML elements with it depending upon the performance needs. It helps in focusing on the graph and the data rather than spending time on the physics and math that makes the layouts possible [24].

In the graph a node represent a web page. Each node also contains further information on resources included in the web page and presented when the node encounters a mouseover. In each node three sub-nodes, scripts, css, and images contain the information on their respective kind. On selecting a particular resource, all the nodes i.e. the web pages using or including that resource is highlighted. This information can be of great help while deciding to make some changes to the resources of a web page, as it will provide the developer a view of what other web pages will be affected by the change.

Apart from the graph based presentation the report also includes a table which provides a text based view of the web pages accessed along with their time of request and the response time. Further two plots for interpreting resource usage is also generated. One plot is drawn on the basis of the resource load time while the other takes the file size into consideration. These plots help in making decisions e.g. In case a given resource is being referred from multiple independent server locations then the server with low response time for a given resource could be chosen to speed up resource load time.

4.3 Results

Results are produced by accessing 13 links under the domain of "http://nitrkl.ac.in". Figure 4.1 shows the image when mouse is hovered on "ModalPopUp.css" resource. The highlighted links show that they also referenced this "ModalPopUp.css" in their Web pages. Figure 4.2 shows the link requested, the time of link request, and the response

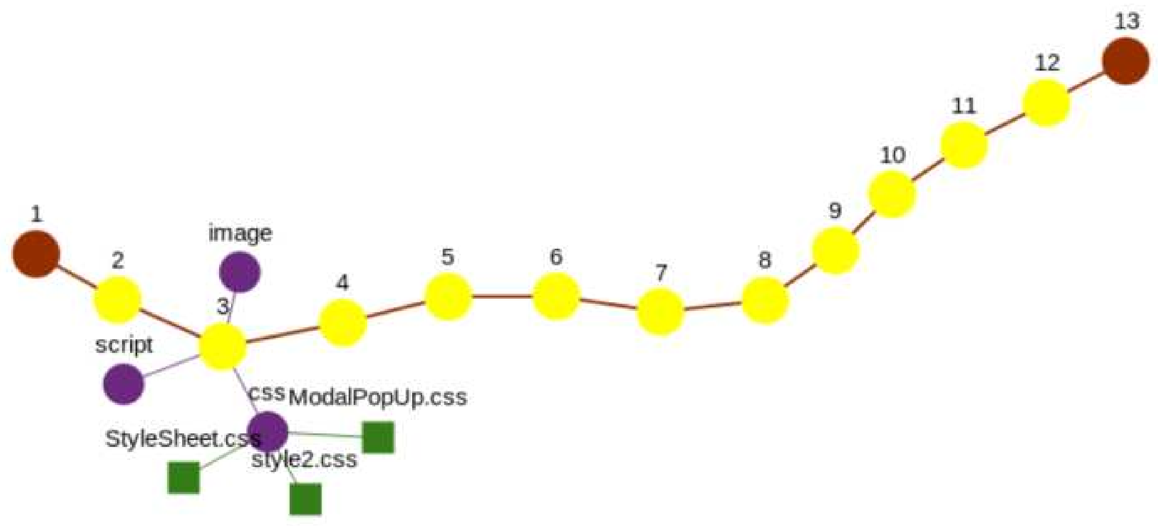
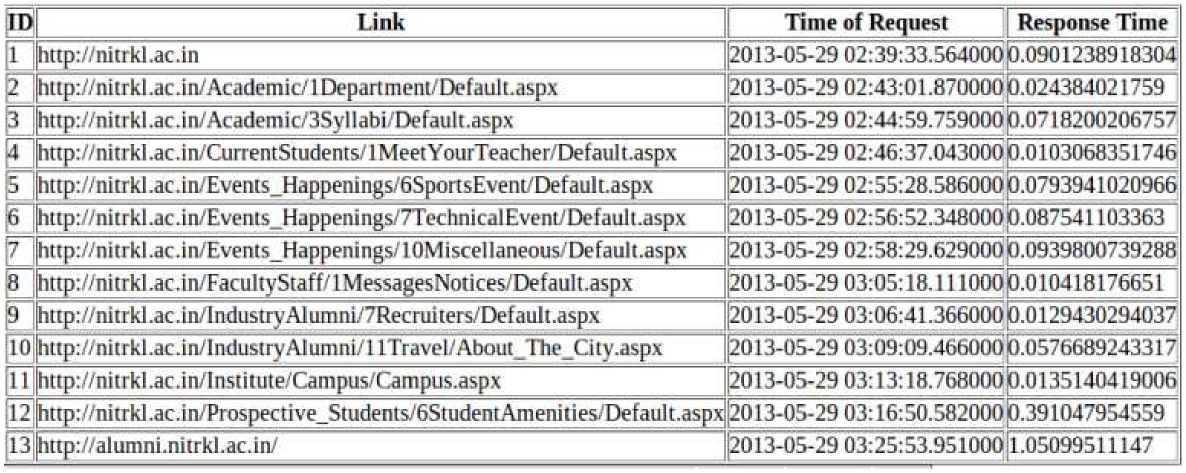


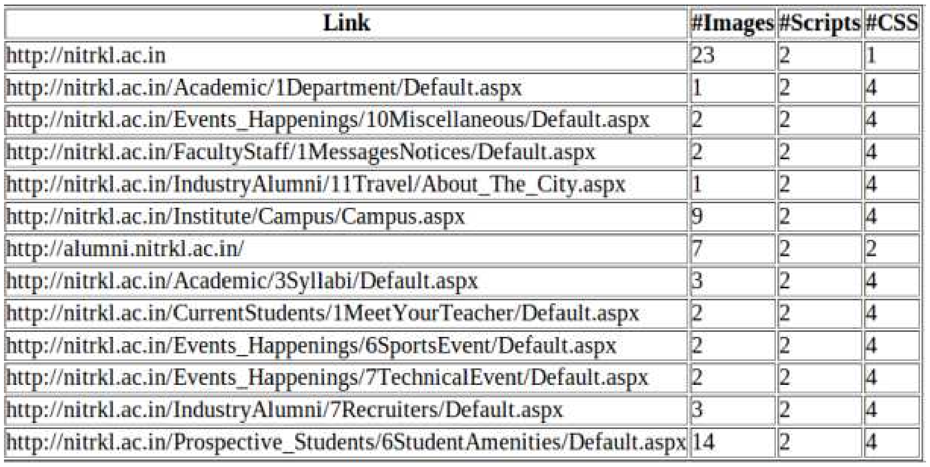
Figure 4.1: Interactive Graph Visualization

time in second. Figure 4.3 shows the link requested along with the number of images, scripts and style sheets it has accessed.



ID	Link	Time of Request	Response Time
1	http://nitrkl.ac.in	2013-05-29 02:39:33.564000	0.0901238918304
2	http://nitrkl.ac.in/Academic/1Department/Default.aspx	2013-05-29 02:43:01.870000	0.024384021759
3	http://nitrkl.ac.in/Academic/3Syllabi/Default.aspx	2013-05-29 02:44:59.759000	0.0718200206757
4	http://nitrkl.ac.in/CurrentStudents/1MeetYourTeacher/Default.aspx	2013-05-29 02:46:37.043000	0.0103068351746
5	http://nitrkl.ac.in/Events_Happenings/6SportsEvent/Default.aspx	2013-05-29 02:55:28.586000	0.0793941020966
6	http://nitrkl.ac.in/Events_Happenings/7TechnicalEvent/Default.aspx	2013-05-29 02:56:52.348000	0.087541103363
7	http://nitrkl.ac.in/Events_Happenings/10Miscellaneous/Default.aspx	2013-05-29 02:58:29.629000	0.0939800739288
8	http://nitrkl.ac.in/FacultyStaff/1MessagesNotices/Default.aspx	2013-05-29 03:05:18.111000	0.010418176651
9	http://nitrkl.ac.in/IndustryAlumni/7Recruiters/Default.aspx	2013-05-29 03:06:41.366000	0.0129430294037
10	http://nitrkl.ac.in/IndustryAlumni/11Travel/About_The_City.aspx	2013-05-29 03:09:09.466000	0.0576689243317
11	http://nitrkl.ac.in/Institute/Campus/Campus.aspx	2013-05-29 03:13:18.768000	0.0135140419006
12	http://nitrkl.ac.in/Prospective_Students/6StudentAmenities/Default.aspx	2013-05-29 03:16:50.582000	0.391047954559
13	http://alumni.nitrkl.ac.in/	2013-05-29 03:25:53.951000	1.05099511147

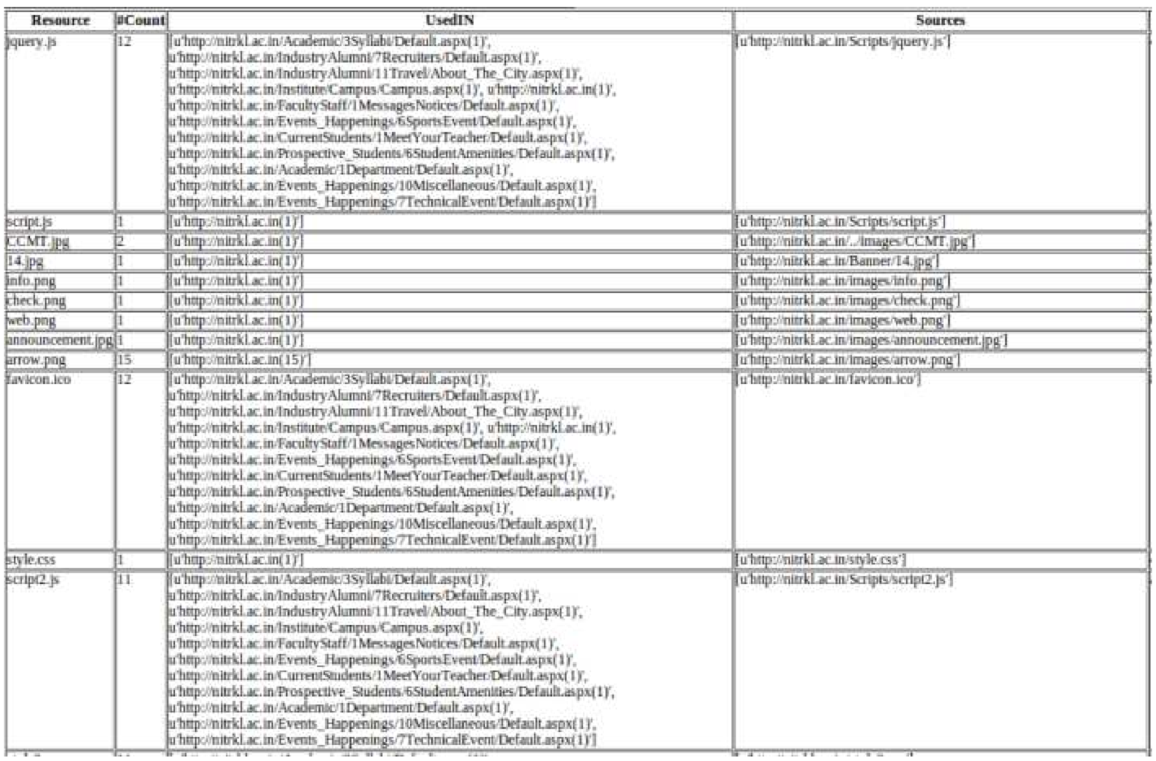
Figure 4.2: Links with Response Time and Time of Request



Link	#Images	#Scripts	#CSS
http://nitrkl.ac.in	23	2	1
http://nitrkl.ac.in/Academic/1Department/Default.aspx	1	2	4
http://nitrkl.ac.in/Events_Happenings/10Miscellaneous/Default.aspx	2	2	4
http://nitrkl.ac.in/FacultyStaff/1MessagesNotices/Default.aspx	2	2	4
http://nitrkl.ac.in/IndustryAlumni/11Travel/About_The_City.aspx	1	2	4
http://nitrkl.ac.in/Institute/Campus/Campus.aspx	9	2	4
http://alumni.nitrkl.ac.in/	7	2	2
http://nitrkl.ac.in/Academic/3Syllabi/Default.aspx	3	2	4
http://nitrkl.ac.in/CurrentStudents/1MeetYourTeacher/Default.aspx	2	2	4
http://nitrkl.ac.in/Events_Happenings/6SportsEvent/Default.aspx	2	2	4
http://nitrkl.ac.in/Events_Happenings/7TechnicalEvent/Default.aspx	2	2	4
http://nitrkl.ac.in/IndustryAlumni/7Recruiters/Default.aspx	3	2	4
http://nitrkl.ac.in/Prospective_Students/6StudentAmenities/Default.aspx	14	2	4

Figure 4.3: Links with Number and Types of Resources

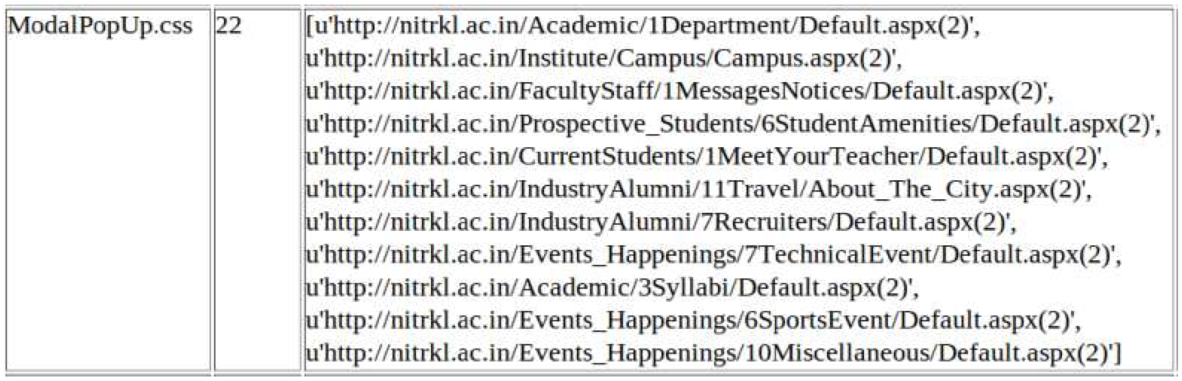
Figure 4.4 shows the resource file, the number of times it has been referenced, the link requests in which it has been referenced along with the frequency of reference, and URI of the resource.



Resource	#Count	UsedIN	Sources
jquery.js	12	[u'http://nitrkl.ac.in/Academic/3Syllabi/Default.aspx(1)', u'http://nitrkl.ac.in/IndustryAlumni/7Recruiters/Default.aspx(1)', u'http://nitrkl.ac.in/IndustryAlumni/11Travel/About_The_City.aspx(1)', u'http://nitrkl.ac.in/Institute/Campus/Campus.aspx(1)', u'http://nitrkl.ac.in(1)', u'http://nitrkl.ac.in/FacultyStaff/1MessagesNotices/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/6SportsEvent/Default.aspx(1)', u'http://nitrkl.ac.in/CurrentStudents/1MeetYourTeacher/Default.aspx(1)', u'http://nitrkl.ac.in/Prospective_Students/6StudentAmenities/Default.aspx(1)', u'http://nitrkl.ac.in/Academic/1Department/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/10Miscellaneous/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/7TechnicalEvent/Default.aspx(1)']	[u'http://nitrkl.ac.in/Scripts/jquery.js']
script.js	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/Scripts/script.js']
CCMT.jpg	2	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/../images/CCMT.jpg']
14.jpg	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/Banner/14.jpg']
info.png	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/images/info.png']
check.png	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/images/check.png']
web.png	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/images/web.png']
announcement.jpg	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/images/announcement.jpg']
arrow.png	15	[u'http://nitrkl.ac.in(15)']	[u'http://nitrkl.ac.in/images/arrow.png']
favicon.ico	12	[u'http://nitrkl.ac.in/Academic/3Syllabi/Default.aspx(1)', u'http://nitrkl.ac.in/IndustryAlumni/7Recruiters/Default.aspx(1)', u'http://nitrkl.ac.in/IndustryAlumni/11Travel/About_The_City.aspx(1)', u'http://nitrkl.ac.in/Institute/Campus/Campus.aspx(1)', u'http://nitrkl.ac.in(1)', u'http://nitrkl.ac.in/FacultyStaff/1MessagesNotices/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/6SportsEvent/Default.aspx(1)', u'http://nitrkl.ac.in/CurrentStudents/1MeetYourTeacher/Default.aspx(1)', u'http://nitrkl.ac.in/Prospective_Students/6StudentAmenities/Default.aspx(1)', u'http://nitrkl.ac.in/Academic/1Department/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/10Miscellaneous/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/7TechnicalEvent/Default.aspx(1)']	[u'http://nitrkl.ac.in/favicon.ico']
style.css	1	[u'http://nitrkl.ac.in(1)']	[u'http://nitrkl.ac.in/style.css']
script2.js	11	[u'http://nitrkl.ac.in/Academic/3Syllabi/Default.aspx(1)', u'http://nitrkl.ac.in/IndustryAlumni/7Recruiters/Default.aspx(1)', u'http://nitrkl.ac.in/IndustryAlumni/11Travel/About_The_City.aspx(1)', u'http://nitrkl.ac.in/Institute/Campus/Campus.aspx(1)', u'http://nitrkl.ac.in/FacultyStaff/1MessagesNotices/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/6SportsEvent/Default.aspx(1)', u'http://nitrkl.ac.in/CurrentStudents/1MeetYourTeacher/Default.aspx(1)', u'http://nitrkl.ac.in/Prospective_Students/6StudentAmenities/Default.aspx(1)', u'http://nitrkl.ac.in/Academic/1Department/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/10Miscellaneous/Default.aspx(1)', u'http://nitrkl.ac.in/Events_Happenings/7TechnicalEvent/Default.aspx(1)']	[u'http://nitrkl.ac.in/Scripts/script2.js']

Figure 4.4: Resources with frequency of Accesses in Different Web Pages

Figure 4.5 shows an interesting scenario that "ModalPopUp.css" resource is referenced in each of the links twice, which is redundant and unnecessary. One reference can be removed from each link. Figure 4.6 and Figure 4.7 show the resources vs. size and resources vs. load time, respectively.



ModalPopUp.css	22	[u'http://nitrkl.ac.in/Academic/1Department/Default.aspx(2)', u'http://nitrkl.ac.in/Institute/Campus/Campus.aspx(2)', u'http://nitrkl.ac.in/FacultyStaff/1MessagesNotices/Default.aspx(2)', u'http://nitrkl.ac.in/Prospective_Students/6StudentAmenities/Default.aspx(2)', u'http://nitrkl.ac.in/CurrentStudents/1MeetYourTeacher/Default.aspx(2)', u'http://nitrkl.ac.in/IndustryAlumni/11Travel/About_The_City.aspx(2)', u'http://nitrkl.ac.in/IndustryAlumni/7Recruiters/Default.aspx(2)', u'http://nitrkl.ac.in/Events_Happenings/7TechnicalEvent/Default.aspx(2)', u'http://nitrkl.ac.in/Academic/3Syllabi/Default.aspx(2)', u'http://nitrkl.ac.in/Events_Happenings/6SportsEvent/Default.aspx(2)', u'http://nitrkl.ac.in/Events_Happenings/10Miscellaneous/Default.aspx(2)']

Figure 4.5: Resource : ModalPopUp.css

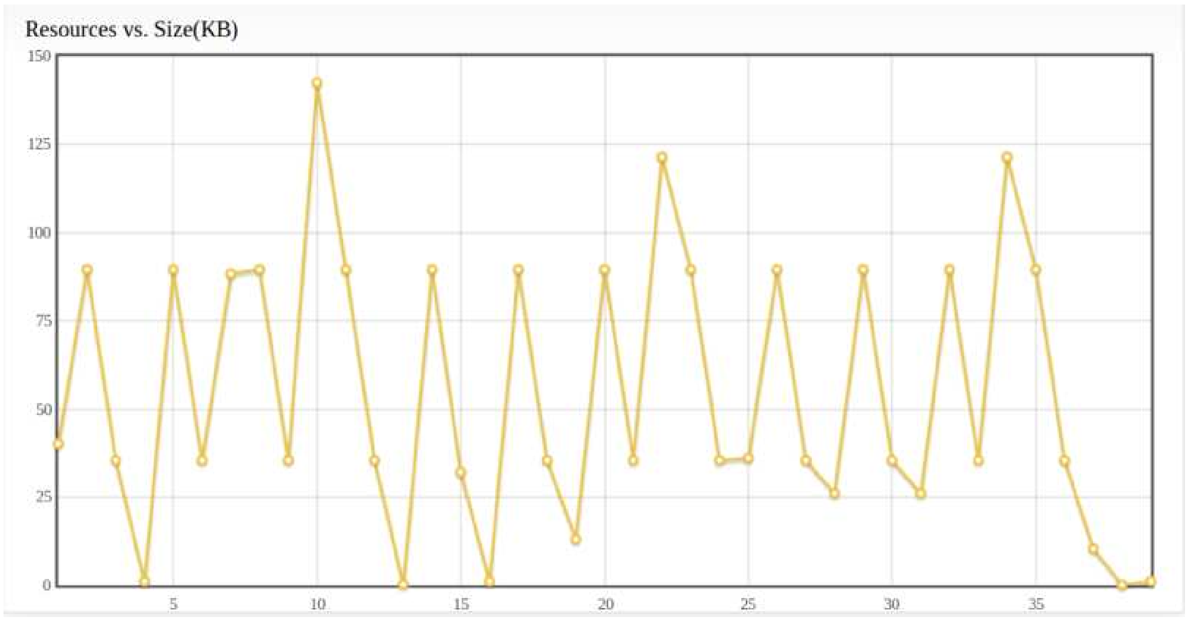


Figure 4.6: Resources vs. Size(in KB)

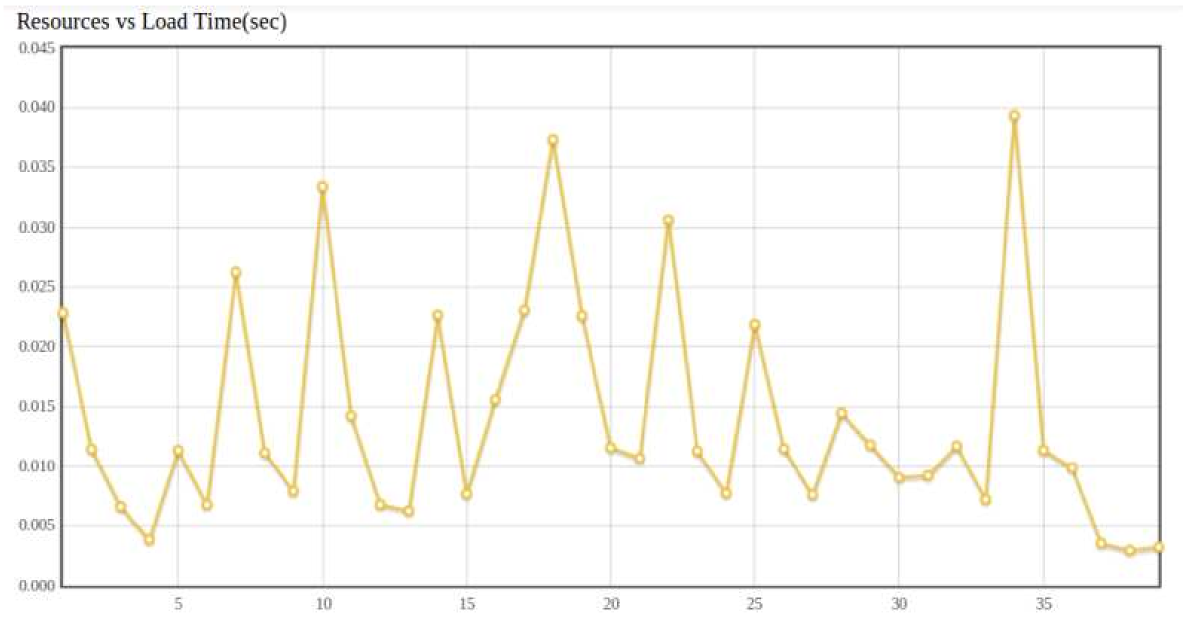


Figure 4.7: Resources vs. Load Time(in sec)

4.4 Conclusion

In this chapter the implementation process has been discussed. The output generated from the monitoring system when subjected to a test environment is also presented and analysed in the Results section. It also includes the graph visualization, and other plots generated by using the monitoring system.

Chapter 5

Conclusion

5.1 Conclusion

To summarize, a scheme to monitor resource usage in web applications is proposed as part of the work. This is in response to the growing complexity and different development practices adopted in web development which makes it difficult to maintain resources efficiently. Many applications available today suffer from poor resource management which results in increased page load time or unintentional removal of the resource itself. Hence the proposed scheme uses a simple approach towards resource monitoring that can aid developers in visualizing the resource usage and the effect on the application when some changes are made to the resources. A working implementation of the scheme was developed and it was observed that for small to medium scale web applications, the scheme can help in some speedup.

5.2 Future Suggestions

The result graph can be populated with details of the resource whenever mouse hovers it. The proposed model can be extended for monitoring other types of resources. An API based resource monitoring for use by developers can also be considered which will allow them to make decisions during the coding step itself.

Bibliography

[1] Mehdi Jazayeri. Some trends in web application development. In *Future of Software Engineering, 2007. FOSE '07*, pages 199–213, 2007.

[2] Feng Liu, Zhenming Lei, and Hui Miao. Web performance analysis on real network. In *Electronics, Communications and Control (ICECC), 2011 International Conference on*, pages 1780–1783, 2011.

[3] Raj Bala Simon and Laxmi Ahuja. Website monitoring: Contemporary way to test and verify. *Global Journal of Enterprise Information System*, 4(1), 2012.

[4] Hongyan Mao, Linpeng Huang, and Minglu Li. Web resource monitoring based on common information model. In *Services Computing, 2006. APSCC '06. IEEE Asia-Pacific Conference on*, pages 520–525, 2006.

[5] Lin Deng, Weifeng Xu, and Stephen Frezza. A resource-based approach to extend uml diagrams for web applications. In *Computer Science and Service System (CSSS), 2011 International Conference on*, pages 103–106, 2011.

[6] Brian Lavoie and Henrik Frystyk Nielsen. Web characterization terminology and definitions sheet. `http://www.w3.org/1999/05/WCA-terms/01`, May 1999.

[7] Nicholas H Lurie and Charlotte H Mason. Visual representation: Implications for decision making. *Journal of Marketing*, pages 160–177, 2007.

[8] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

[9] Carl Zetie. The rise of rich internet applications. *Trends*, April 2006.

[10] Keynote. Measuring and monitoring web 2.0 applications. New York, December 2009.

[11] J. Sergio Zepeda and Sergio V. Chapa. From desktop applications towards ajax web applications. In *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*, pages 193–196, 2007.

[12] Bernard Kohan. Guide to web application development. `www.comentum.com/guide-to-web-application-development.html`, April 2013.

[13] Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, Albert Greenberg, and Yi-Min Wang. Webprophet: Automating performance prediction for web services. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 10–10. USENIX Association, 2010.

[14] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. Demystifying page load performance with wprof. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 473–486. USENIX Association, 2013.

[15] Jukka Palomaki. *Web Application Performance Testing.* PhD thesis, University of Turku, December 2009.

[16] Steve Souders. High-performance web sites. *Communications of the ACM*, 51(12):36–41, 2008.

[17] Zhang Xu and Dong Yan. Designing and implementing of the webpage information extracting model based on tags. In *Intelligence Science and Information Engineering (ISIE), 2011 International Conference on*, pages 273–275. IEEE, 2011.

[18] Corey Goldberg. Automated web/http profiler with selenium and python. `https://code.google.com/p/selenium-profiler`, 2011.

[19] Pingdom website speed test. `http://tools.pingdom.com/fpt`, April 2013.

[20] web.py tutorial. `http://webpy.org/docs/0.3/tutorial`, 2013.

[21] MongoDB. Introduction to mongodb. `http://www.mongodb.org/about/introduction/`, 2013.

[22] 10Gen. What is nosql? `http://www.10gen.com/nosql`, 2013.

[23] Beautiful soup. `http://www.crummy.com/software/BeautifulSoup/`, May 2013.

[24] arbor.js introduction. `http://arborjs.org/introduction`, March 2013.