# Analysis and Implementation of an Ajax-enabled Web Crawler

Li-Jie Cui[1], Hui He[2] and Hong-Wei Xuan[1]

*[1]School of Software and Engineering,*
*Harbin University of Science and Technology, Harbin, China*
*[2]Harbin Institute of Technology, Harbin, China*
*andyclj1977@163.com, hehui@hit.edu.cn, henryxuan@hotmail.com*

***Abstract***

*This paper analyzes a web crawler for the Web 2.0 network, which presents new challenges. A comprehensive overview on various programs and strategies is presented, which includes the design and realization of Ajax reptiles. Experimental verification indicates that an Ajax Crawler can effectively obtain Ajax dynamic pages. The proposed Ajax Crawler is then compared with common Ajax reptiles in terms of their download speeds.*

***Keywords:*** *Dynamic web pages, Ajax, Web 2.0, web crawler*

## 1. Introduction

Web 2.0 is a general designation for a new class of Internet applications relative to Web 1.0. User access to information through the browser is a main feature of Web 1.0. Web 2.0 is more focused on user interaction, and both website content viewers and the manufacturers of the site content are considered users. Internet users are no longer solely readers; they are authors who “surf” the Web, but can become “wave makers.” Thus, the user participation develops from the simple ability to “read” to “write” until it evolves into the ability to “build.” Thus, users shift from being a passive receiver of information on the Internet to a proactive contributor of information, thereby making the system more user-oriented.

With the rising popularity of Web 2.0 in web development, more websites use Ajax technologies. The application of this technology has improved the user experience, generated several new concepts, and enhanced the design of the Web user interface. Simultaneously, technology can dynamically change the content and the traditional structure of the web page. The web crawler uses static web pages to produce less content during page rendering. However, this dynamic content tremendously challenges the web crawler design. Thus, these web crawlers cannot access the full text of the presented web page.

This paper supports the development of an Ajax web crawler that supports Ajax and can execute custom tasks. Thus, the acquisition of web information becomes highly efficient. Section 2 describes the state-of-the-art for Ajax pages and previous crawl studies that were performed both locally and abroad. Section 3 introduces the design and concrete realization of reptiles that support Ajax. Section 4 verifies the feasibility of applying Ajax reptiles through several experiments.

This paper designed a web crawler, which can successfully crawl Ajax dynamic web pages, and compared with ordinary Ajax reptiles in the download speed and the number of downloaded pages.

## 2. Related Work

To date, mainstream search engines like Google and Yahoo do not index dynamic pages. Thus, users cannot use search engines to find dynamic page content.

Search engines that support Ajax are still in the research stages. Luo [1] designed a search engine that crawls the Ajax page; this program contains JavaScript code files to obtain pages based on protocol, to directly design and implement script interpreters, to simulate the behavior of the browser while executing the script codes, and to implement control over the conversion of a page state. Xiao [2] used the same approach in a later study. Zeng and Li [3] built a program with a hierarchical model based on the slicing algorithm to solve the orderly execution problem of JavaScript. Frey [4] and Matter [5] improved the existing version of Rhino [6], an open source JavaScript interpreter to implement script execution and state transition. However, completely reproducing the script interpreter is time consuming. Although the Rhino project has been maintained for several years, numerous script codes are not working properly. Therefore, other studies have attempted to take advantage of the embedded browser component to implement the rendering and script execution of Ajax pages, thereby achieving the intended automatically converted state. Wang [7] and Jin [8] reportes studies similar to those of Frey [4] and Matter [4], where web crawler technology was evaluated based on script language analysis using the open-source JavaScript engine SpiderMonkey and Rhino to parse JavaScript dynamic pages, to extract then execute JavaScript codes, and to obtain URL address that contain these pages, thereby completing the crawling task.

Frey and Matter extended the Cobra tool set [9], as well as implemented dynamic loading and parsing of HTML codes, until they accessed the DOM structure tree formed by the internal state to obtain the data content of the state. Similar with the state transitions, their study makes the full use of existing technology and the mainstream acquisition mode of state content by using the embedded browser as the Ajax running container, interacting with the external interface provided by the browser and the page DOM tree, and obtaining the full data content [10–13].

A relatively simple method of conversion control is to use the event filter mechanism because each event is triggered one by one after filtering. To improve the efficiency, Duda [14] and Xia [12] proposed rules to support user-defined rules and to introduce the receiver and the refusing rules. Thus, the state transitions can only occur using a finite set that meets the conditions. Mesbah [15] used three different ways to control the state transitions, namely, fully automatic scanning, HTML element annotation, and field-oriented manual configuration. Matter [16] reported a heuristic crawling strategy to avoid the same pages from different paths.

## 3. Ajax Crawler reptile design solution and achievements

More websites use Ajax technology because of the popularity of Web 2.0. The Rhino engine is used in this study to support Ajax.

### 3.1. JavaScript engine Rhino

Rhino is a JS scripting engine using pure Java language. Rhino is used in a Java program to provide end users with scripted capabilities [17]. Rhino contains a JavaScript code parser, compiler, and code debugging modules.

*Script compiler module.* The input of the compiler is the JavaScript code, whereas its output is a JavaScript data object. The JavaScript structure contains the byte code, comments,

a string pool, and the data, as well as an identifier. JavaScript likewise contains objects and functions, among others. Functions are a period of nested JavaScript codes [18]. Compilers consists of three components, namely, the lexical scanner of random logic used to create the AST recursive descent parser of tree-walking, the code generator which is a compilation process completed by function Main, and the processSource () that compiles the input character strings or JavaScript codes into the stream file. During the compilation process, all of the variables, symbols, and commands are parsed by the lexical parser, identified using specific internal symbols, and placed into the corresponding data stack. Finally, the data will be returned with a tree structure, which is easier for the Rhino engine to compile. In the case of less semicolon or assignment expression, semantic and lexical feedback mechanisms eliminate ambiguity. The compiler does not have error correction. Therefore, when an error is encountered, the compiler halts immediately [19]. The compiler adds the source notes information into the compiled script structure so they can be used when users call the function toSource () to decompile.

*The script interpreter module.* Similar to other JavaScript engines, Rhino's interpreter is a single-threaded huge cycle function, which can explain an instruction in bytecode [20]. This huge function uses a switch statement to execute different bytecodes, before it jumps into a different execution segment of the codes. If a piece of JavaScript code calls another section of the same code, the engine will usually only use the JavaScript stack space and the interpreter is executed sequentially [21]. However, if the JavaScript code calls the java code, the java code then calls the JavaScript code, thereby causing reentrant problems with the interpreter. Fortunately, this function is reentrant. Various states needed for the interpreter will pass the parameters when the interpreter function enters. A vast majority of the states are stored in the data structure of context [22]. Therefore, the first parameter of all public API interfaces for Rhino, as well as most of the function interfaces, is a context pointer.

### 3.2. Ajax reptile architecture design

The reptile system is divided into two parts. The first part is the pre-processing stage, which is responsible for cleaning the URL information to remove unnecessary crawled URLs similar to the function of the filter. The second part is the actual crawling function of the web page. To take full advantage of the available resources because downloading is time consuming, we used a multi-threaded crawl.

The crawling stage obtains web pages, resolves the internal links of these web pages, and executes pretreatment. Not all the internal links of a web page need to crawl. The internal links of web pages may contain previously grabbed and duplicated URLs, which prohibits crawling in the robot protocol. The clean URLs are URL warehouses, where the URLs that need to crawl are stored.

Web deduplication Management connects with Link filter, which includes Web deduplication, HTTP Protocol control, URL Document format control and Robots control. Among them, Robots control is managed by Robots Document management. They are sequentially executed. And then, it enters the next process through Clean URLs. The next process contains DNS Parsing, Get Request, Original page, Page processing and Link crawl. The five steps constitute the Multi-threaded crawl. DNS Parsing is managed by DNS Cache. Ajax Page obtained by Original Page enters Ajax Engine, which exports Static Page to the Page processing.

The reptile architecture diagram is shown in Figure 1.

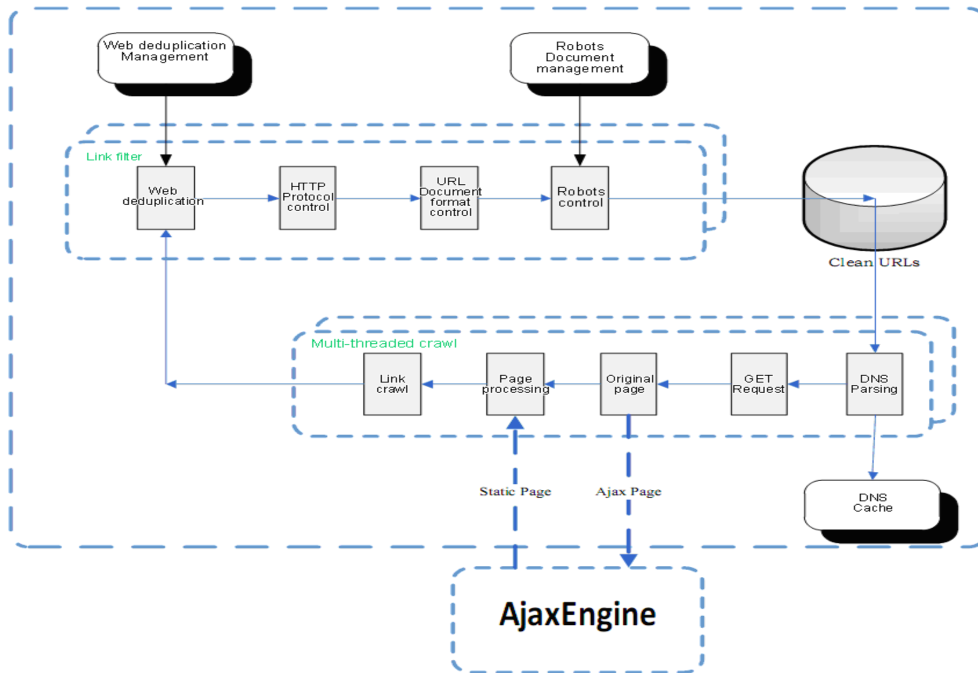



**Figure 1. Ajax Crawler Architecture Diagram**

### 3.3. Ajax Engine design

The Ajax Engine provides support for Ajax in the web pages. The engine can parse the js code in a web page, and obtain the dynamic content. The diagram showing the Ajax Engine architecture is shown in Figure 2.

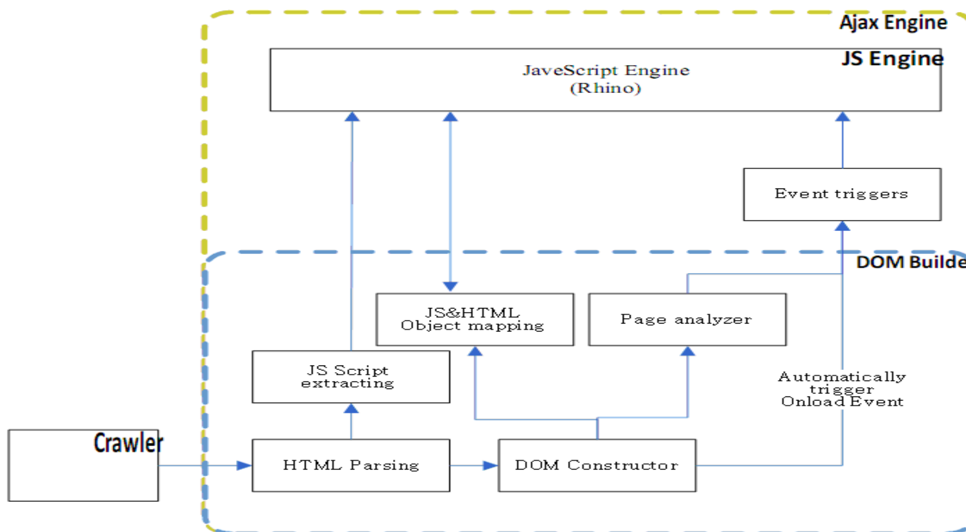



**Figure 2. Ajax Crawler Engine Architecture Diagram**

From the figure, the Ajax engine has three components. An Ajax page is analyzed in the following order.

(1) The crawler obtains the page needed to be crawled by HTTP request. This page contains the Ajax code, in which no real content exist.

(2) The DOM Builder analyzes the page, builds the DOM tree, and extracts the JS code, which consequently triggers corresponding events.

(3) Ajax codes are placed into the JavaScript engine according to the JS objects to be executed. During execution, HTML objects are mapped to modify the HTML object.

(4) The regrouping of the implementation results generates new page content that returns to the crawler.

## 4. Results of Experimental Data

### 4.1. Throughput experiments

Throughput experiments test the number of Ajax Crawlers and the relationship between the number of start threads and download speed of each Ajax Crawler. If the start of the Ajax Crawler or the number of threads of each Ajax Crawler is too few, the system cannot take full advantage of the resources of the computer. Otherwise, the efficiency is reduced because of the bandwidth and data competition. The test machine is a common PC, whose hardware configuration and software configuration are shown in Table 1.

**Table 1. Server Configuration for Ajax Crawler**

| | |
|---|---|
| CPU | 2 * AMD Athlon™II X2 215 Processor 2.70 GHz |
| Network card | 100 Mbps |
| Memory | 2.00 GB |
| Operating system | Windows 7 |
| Java environment | JRE 1.6.0_24 |

The Ajax Crawler test task was the TianYa blog. Each Ajax Crawler and the number of threads in the said crawler were changed. The measured data is shown in Table 2. (unit: number of pages/10 min).

**Table 2. Test Results of Ajax Crawler**

| | Number of threads | | | |
|---|---|---|---|---|
| Ajax Crawler | 1 | 2 | 3 | 4 |
| 1 | 467 | 879 | 1346 | 1301 |
| 2 | 953 | 1029 | 1634 | 1529 |
| 3 | 1369 | 1473 | 1679 | 1723 |
| 4 | 1321 | 1493 | 1709 | 1634 |

The data is plotted as a line chart, as shown in Figure 3.

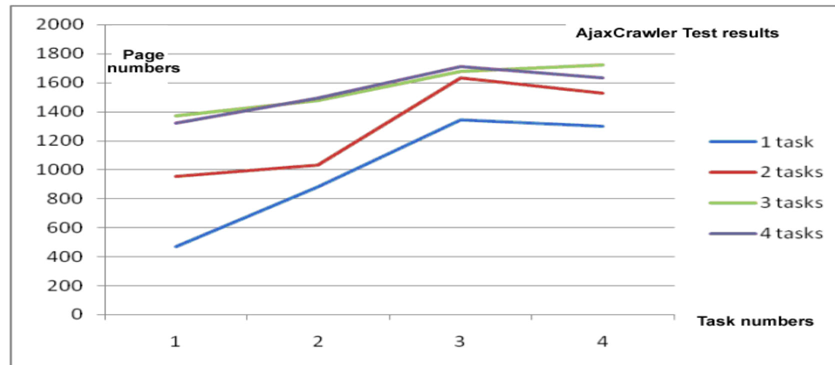



**Figure 3. Blog Crawler Test Results**

From Table 2 and Figure 3, when we select three tasks and three threads within each task in the current computing environment and network environment. The operation efficiency of the proposed system is superior, which can download 1679 pages on the average every 10 minutes.

### 4.2. Ajax Crawler comparison experiments with ordinary reptiles

In this experiment, the NetEase reviews become the test object to analyze the efficiency of the Ajax Crawler. Given that unit quantitative indicators are no longer reported as the number of pages/10 minutes, Kb/second is selected because ordinary crawler cannot crawl dynamic content. The test results are listed in Table 3, which have been plotted as line charts (Figure 4).

**Table 3. Comparison of Ajax Crawler and Web Crawler**

| Number of tasks | Ajax Crawler | Web Crawler |
|---|---|---|
| 1 | 6.3 | 8.53 |
| 2 | 9.12 | 11.76 |
| 3 | 11.13 | 16.2 |
| 4 | 12.5 | 19.5 |

Regardless of the concurrent execution of several tasks, the Ajax Crawler is slower than the corresponding ordinary Web Crawler in terms of their download speed (Table 3 and Figure 4). Aside from the experimental and environmental causal factors, the Ajax Crawler could parse to Ajax script and update the operation of the DOM tree. When closing off the support of a dynamic page, the speed of Ajax Crawler has little difference with the ordinary web crawler. The two lines are nearly parallel in Figure 4 to show that the speed ratio of two kinds of reptiles are almost equal. The accelerated increasing trend of the Ajax Crawler starts slowly, which implies that the Ajax Crawler needs more computing resources.

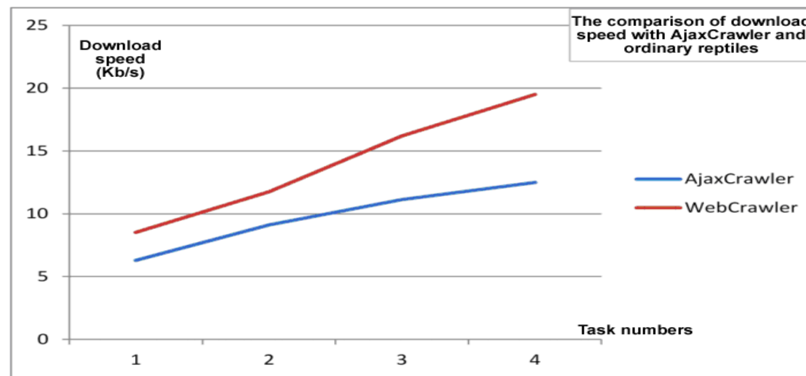



**Figure 4. Comparison of download speed with Ajax Crawler and Web reptiles**

By contrast with the download pages, the difference between the two crawlers is more than ten times, whereas their download speed is relatively slow, which further illustrates that the dynamic pages contain rich content.

## 5. Conclusion

This article designed a web crawler, which can successfully crawl Ajax dynamic web pages. Compared with ordinary Ajax reptiles, the download speed of Ajax Crawler is slower because the Ajax script parsing and the DOM tree updating are relatively time-consuming. However, the number of downloaded pages by the Ajax crawler will be more than ten times that of ordinary reptiles, and the content obtained by crawling is richer than that obtained by ordinary Ajax reptiles.

## Acknowledgements

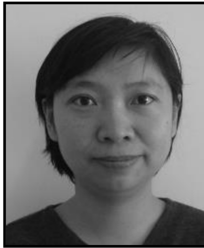
This work was supported by the national high technology research and development plan(2010AA012504、2011AA010705) and the national key basic research and development plan(G2011CB302605、2007CB311101) and the National Natural Science Fund(61173145).

## References

[1] Bing Luo, "Design and Implementation of Internet search engine crawlers supported AJAX', HangZhou. ZheJiang University, **(2007)**.
[2] Zhuo-Lei Xiao, "Research on the search engine based on the AJAX technology",.WuHan, Wu Han University of Science and Technology.
[3] Wei-Hui Zeng, Miao Li, "The Ajax framework network reptiles technology based on JavaScript slices", Computer Systems & Applications, vol. 18, no. 7, **(2009)**.
[4] G. Frey, "Indexing Ajax Web Applications", Zurich: Swiss Federal Institute of Technology Zurich, **(2007)**.
[5] R. Matter, "Ajax Crawl: Making Ajax Applications Searchable", Zurich: Swiss Federal Institute of Technology Zurich, **(2008)**.
[6] Mozilla. Rhino: JavaScript for Java, [2009-03-22].
[7] http://www.mozilla.org/rhino/.
[8] Ying Wang, Mian-Quan Yu and Seng-Tao Li, "JavaScript engine applied in dynamic web collection technology", Computer Applications, vol. 24, no. 2, **(2004)**.
[9] Xiao-Ou Jin, Bao-Yan Zhong and Xiang Li, "Analytical Study and Implementation of JavaScript dynamic page based on the Rhino", Computer Technology and Development, vol. 18, no. 2, **(2008)**.
[10] Cobra. Java HTML Renderer & Parser, [2009-01-19].
[11] http://lobobrowser.org/cobra.jsp.

[12] A. Mesbah, E. Bozdag and A. Van Deursen, "Crawling Ajax by Inferring User Interface State Changes", Proceedings of the 8th International Conference on Web Engineering, Yorktown Heights, NJ., Washington, DC, USA, IEEE Computer Society, **(2008)**, pp. 122-134.
[13] H. Guo, Y. -L. Lu and J. -H. Liu, "Ajax crawling algorithm based on state transition diagram, vol. 26, no. 11, **(2009)**.
[14] T. Xia, "Extracting Structured Data from Ajax Site", Proceedings of 2009 International IEEE Workshop on Database Technology and Applications, Wu han, China, **(2009)**, pp. 259-262.
[15] X. -J. Yuan, "Research and implementation based on protocol-driven and event-driven comprehensive focused crawler", Chang Sha, National Defense Science and Technology University, **(2009)**.
[16] C. Duda, G. Frey and D. Kossmann, "Ajax Search: Crawling, Indexing and Searching Web 2.0 Applications", Proceedings of the VLDB Endowment Archive, vol. 1, no. 2, **(2008)**, pp. 1440-1443.
[17] J. K. Lee, S. M. Nam and T. H. Cho, "An Improved Method for Probabilistic Voting-based Filtering using Blacklists in Sensor Networks", IJFGCN, vol. 5, no. 3, **(2012)** September, pp. 1-10.
[18] Y. Han, S. -J. Seok, W. -C. Song, D. Choi and J. -W. Huh, "Detection of Greedy Nodes in Wireless LAN through Comparing of Probability Distributions of Transmission Intervals", IJMUE, vol. 8, no. 1, **(2013)** January, pp. 175-184.
[19] A. A. A. Ssaed, W. M. N. W. Kadir and S. Z. M. Hashim, "Metaheuristic Search Approach Based on In-house/Out-sourced Strategy to Solve Redundancy Allocation Problem in Component-Based Software Systems", IJSEIA, vol. 6, no. 4, **(2012)** October, pp. 143-154.
[20] J. Zhang, Y. Cui and Z. Chen, "SPA: Self-certified PKC-based Privacy-preserving Authentication Protocol for Vehicular Ad Hoc Networks", IJSIA, vol. 6, no.2, **(2012)** April, pp. 409-414.
[21] L. Yu, "Understanding component co-evolution with a study on Linux", Empirical Software Engineering, vol. 12, no. 2, **(2007)**.
[22] Q. Guan-qun, Z. Li, Z. Lin and P. Lew, "Modeling Method and Characteristics Analysis of Software Dependency Networks", Computer Science, vol. 11, no. 35, **(2008)**, pp. 239-243.

## Authors



**Li-Jie Cui** is a lecture in school of software, Harbin University of Science and Technology, China. She was born on February 1977. She achieved her Master degree in software engineering in 2005 at Harbin Institute of Technology. Her research emphasizes on information technology, software engineering and network security.

**Hui He** received the B.S., M.S. and Ph.D. degree in computer science from Harbin Institute of Technology, Harbin, China. Since September 1999, she has been with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China, where she became an Associate Professor in October 2007. Her research interests include network computing, network security.

**Hong-Wei Xuan** is a lecture in School of Software, Harbin University of Science and Technology, China. He achieved his Master degree in Department of Computing, University of Bradford in 2005. His research emphasizes on software engineering and natural language processing.