ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 204-208
doi: 10.18178/wcse.2017.06.035

A Dynamic Partitioning Approach for Ajax Applications

Peng Li[1+]

[1]Beijing Institute of Technology, 5 South Zhongguancun Street, Haidian District, Beijing, China

Abstract. This research investigates programming language and middleware abstractions which offer an alternative approach in Ajax application development, for partitioning program logic specifically with coordination between client and server components that has flexibility in mapping components to physical locations. I work on a middleware and JavaScript interpreter that provides support for Ajax application partitioning. To evaluate the performance of the customized interpreter, I evaluate a well-known Ajax application called Java Pet Store to prove that the customized interpreter provides low-performance overhead.

Keywords: Programming language, Ajax programming, program partitioning, Web development.

1. Introduction

Ajax is a group of Web development techniques, for example JavaScript and XML, used to create rich Internet applications. Currently, Ajax is playing an important role in enhancing the usability and user interaction of Web applications. The programming of Ajax applications is becoming more complicated because Ajax applications move a large portion of application logic to the client's Web browser [1]. So Ajax developers have to deal with code on both client and server, and handle their communication. Traditionally, research on application partitioning has been used to address these problems.

For an Ajax application, an optimal partitioning configuration is not always fixed but sometimes depends on the different client platforms and execution environments; this will create problem for the traditional static partitioning approaches. For example, when a new browser which supports faster JavaScript execution is released, for example, Google Chrome, a developer may want to move more application logic to the client side for scalability or faster response. However, when a mobile phone user needs to access this Web site, a developer may want to move less application logic to the mobile phone browser for the performance consideration. Putting some application logic on the client side will avoid server load and client-server communication latency. This example demonstrates the kinds of decisions that need to be made for Web application partitioning. The answer is not "black and white" but depends on the context, for example, execution environments and capabilities supported by different clients [2].

The distribution of the code between the client and server requires more developmental effort to build and maintain than traditional Web applications. This is because of a classic problem in distributed systems software called the *Location Dependency Problem*: in the case that one module, A, depends on another module, B, then the implementation for A must be embedded with extra knowledge of not only B's interface, but also its physical location. The Location Dependency Problem even gets more complicated when a new feature is added into the existing system, developers always need to consider which part of the new feature should be on the server and which part of the new feature should be on the client. Later on, this configuration could be changed and some parts of the feature could be moved between the client or server for different reasons. These design changes in the architecture can induce crosscutting changes to implementation units unless the architecture had carefully accounted for the potential of those design decisions to change.

+ Corresponding author. Tel.: + 8613910468787.
E-mail address: lipeng360@bit.edu.cn.

2. Motivating Example

In order to motivate my approach, I use an example from an existing open-source Web application called Java Pet Store 2.0 [3] (henceforth, JPS). This online pet store allows the end-user to enter new seller information for an available pet. Here I use the example of the "information about yourself" panel, from which the end-user can enter the seller's information.

Figure 1A shows an informal, high-level data-flow diagram representing the information flow of information submission between client and server (ignoring Figure 1B for now) which is managed by the Controller layer. Once the information is submitted (User's Input), then all the information will be verified by a client side validation function (Client Side Verification); which is written by using JavaScript to provide a fast reaction of input verification. Then, the information is transferred to the server, and the seller's address will be converted to geographic coordinates by using a third-party Web service called Yahoo Geocode, which is used to convert the address to geographic coordinates. Next, the information will be validated again by the Server Side Verification for security reasons. Finally, the verified information and the geographic coordinates will be stored into database (StoreDB).

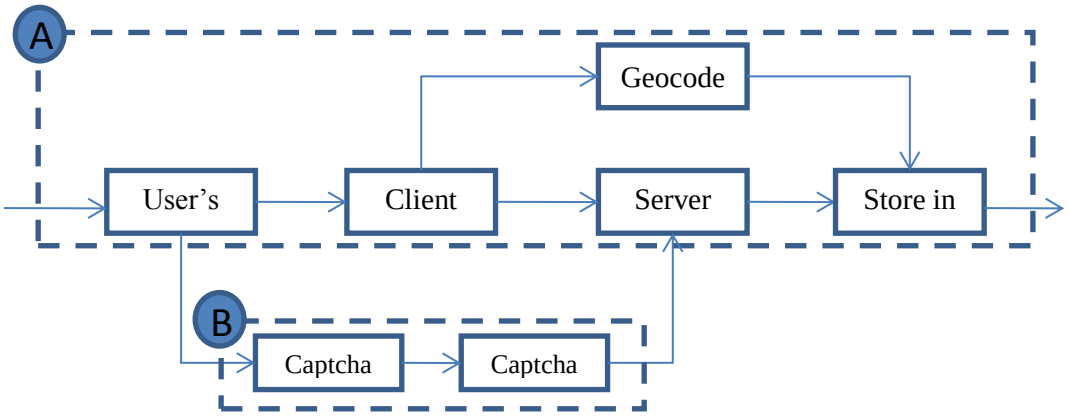

Fig. 1: The components in A represent the original seller information input feature in JPS. Pipeline B represents the Captcha feature data-flow.

The Geocode component (top of Figure 1) is currently implemented on the server-side. However, in some projects [4], this conversion component is used on the client-side for scalability reasons, which potentially makes a different client-server partitioning configuration. In the traditional implement approach, to change to this new partitioning configuration, the dataflow has to be rearranged by modifying not only the dependencies between modules that interact directly with the Geocode component, but also the many components in the JPS responsible for network message passing. To make this change, in the real application, the developers need to change many modules for JPS. According to my investigation, this modification requires crosscuting changes on 17 files in both Java and JavaScript.

3. My Research and Revised Example

To address this problem, I propose to provide a dynamic application partitioning approach to partition an Ajax application by using a custom JavaScript dialect and interpreter. In this case, my interpreter will dynamically partition an Ajax application for different clients who have different platforms and environments and require different partitioning configurations. The JavaScript dialect will be almost identical to standard JavaScript syntactically, but will provide out-of-order execution for some statements to provide optimization of network communication. This style of execution semantics is common for some parallel programming languages [6] and database query languages [10]. As JavaScript is being used more and more for concurrent network programming and Web service-based data composition, I believe this change in execution-style reflects the target domain appropriately. By completing this research I hope to liberate developers from writing the tedious communication logic for all the possible combinations of different locations of components. Furthermore, my framework will provide an optimized network communication policy to minimize time consuming requests from client to server.

3.1. Pet Store Example Revisited

To address the Location Dependency Problem, I show how a developer could move the Geocode functionality to the client side. Using my approach, this task can be done by writing a new c_Geocode function and replacing the c_Geocode function call with a new c_Geocode function call in the coordinator script.

a

```
1.      x = c_Input();
2.      captcha = c_getCaptcha();
3.      c_ClientVerif(x);
4.      addr = getAddress(x.str1, x.city, x.state, x.zip);
5.      pts = s_Geocode(addr);
6.      pricex = getPrice(x.price);
7.      addr = new Addr(x.str1, x.city, x.state, pts.lat, pts.long);
8.      contactInfo = new Seller(x.firstName, x.lastName, x.email);
9.      item = new Item(x.product, x.name, x.desc, pricex,
                                        addr, contactInfo);
10.  s_ServerVerif(item);
11.  c_CaptchaCSVerif(captcha);
12.  s_CaptchaSSVerif(captcha);
13.  s_StoreDB(item);
```

b

```
1.  pts = s_Geocode( "2100 Main Mall, Vancouver, BC" );
2.  addr = new Addr(.., .., .., pts.lat, pts.long);
3.  item = new Item(.., .., .., .., addr, ..);
4.  s_ServerVerif(item);
5.  s_CaptchaSSVerif( "QRM7A" );
6.  s_StoreDB(item);
```

Fig. 2: The coordinator script for the JPS example. Part (a) presents the original coordinator script; part (b) presents the residual script after being evaluated on the client middleware. Code related to the Captcha feature is highlighted. Elided arguments are residual constant values, not shown to simplify presentation.

Figure 3 presents a different residual script after processing by the client middleware, if the Geocode functionality is on the client side (as compared to Figure 2b). Lines 1-3 from Figure 2b are gone because they are now evaluated on the client side along with the new geocode function. Notice for example that in the first case, the address string "Main Mall …" needed to be sent from the client to the server, since the geocoder was on the server. Now the address string is no longer required to be passed across the network, and as illustrated in Figure 3, my approach is intended to detect this and account for it automatically. This shows how my proposed middleware and language is supposed to automatically detect what code was evaluated in what location and provide for flow of data across the network where necessary.

```
1.  s_ServerVerif(item);
2.  s_CaptchaSSVerif( "QRM7A" );
3.  s_StoreDB(item);
```

Fig. 3: A different residual script compared to the Figure 2a when the Geocode functionality is on the client side.

4. Performance Requirement and Experimentation

To explain the performance overhead, I will first see how bad the performance could be if a regular JavaScript interpreter is used to interpret this coordinator script (Figure 2a). It is important to acknowledge that a client-side JavaScript developer would never write a script in this style, because they would know it would result in poor performance. Instead, they would use a less natural coding style to ensure good performance. I am hoping to demonstrate in my paper that both can be achieved simultaneously.

Supposing the script in Figure 2a is executed on a browser, and the Geocode function is on the server side. In this case, a regular interpreter will issue four remote function calls, one for each server function. When use my JavaScript interpreter, the client middleware will attempt to evaluate as many statements as

possible in Figure 2a. So, line 1, 2, 3, 4, 6, 8 will be evaluated because they do not have a dependency on the server side execution. Next, a residual script (Figure 2b) is sent to the server. Finally, the server evaluates all the left statements. We can see there would only be one communication necessary instead of four. I have evaluated message passing for the coordinator script which will be interpreted by using a regular JavaScript interpreter (the number of message passing is 67631) and message passing for the coordinator script which will be interpreted by using my interpreter (the number of message passing is 26762). Based on the result, the amount of message passing is almost triple of message passing in the first situation.

5. Related Work

Some other research is based on a dynamic application partitioning approach, e.g. mobile objects [13], but it does not address the network usage optimization problem. This research argues that the choice of placement for components cannot always be achieved offline. That research makes the point that the location where some components could be placed is context-dependent. From the perspective of [17] (as in the previous paragraph), these components could not be said to be fixed or flexible (one might say they were fixed for a specific context). Since context cannot be determined until run-time, the static approach to message optimization is not applicable. Furthermore, research on dynamic application partitioning, did not address optimization of message passing. Combining the motivation to place components dynamically, and the motivation to optimize message passing, I provide a dynamic partitioning approach which still supports optimized network communication.

6. Acknowledgements

This project is funded by Frontier and interdisciplinary innovation program of Beijing Institute of Technology.

7. References

[1] Livshits, V. B. and Kiciman, E.: Doloto: code splitting for network-bound web 2.0 applications. In ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2008.

[2] Wohlstadter, E., Li, P. and Cannon, B.: Metro: Web Service Mashup Middleware with Partitioning of XML Pipelines. In the International Conference on Web Services, 2009.

[3] Sun Microsystems. Java Pet Store 2. [Online] http://java.sun.com/developer/releases/petstore/.

[4] A framework for rapid integration of presentation components. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., and Matera, M. 2007. International Conference on the World-Wide Web.

[5] Script InSight: Using Models to Explore JavaScript Code from the Browser View. Li, P. and Wohlstadter, E. International Conference on Web Engineering.

[6] Fortress language, http://research.sun.com/projects/plrg/, Visited 9/28/2009

[7] *Mashups: The new breed of Web app.* Merrill, D. IBM Developer Works.

[8] *Mashup Styles*. Ort, E., Brydon, S. and Basler, M. 2007. Sun Developer Network.

[9] Stuart, M. Java GUI Builders. [Online] http://www.fullspan.com/articles/java-gui-builders.html.

[10] XQuery, http://www.w3.org/TR/xquery/, Visited 9/28/2009

[11] W3C. XProc: An XML Pipeline Language. [Online] http://www.w3.org/TR/xproc/.

[12] Apache. Cocoon. [Online] http://cocoon.apache.org/.

[13] Tilevich, E. and Smaragdakis, Y.: J-Orchestra: Automatic Java Application Partitioning. In European Conference on Object Oriented Programming, 2002.

[14] Architecture recovery of web applications. Hassan, A. and Holt, R. 2002. International Conference on Software Engineering.

[15] Using aspectC to improve the modularity of path-specific customization in operating system code. Coady, Y., Kiczales, G., Feeley, M. J. and Smolyn, G. 2001. ACM SIGSOFT International Symposium on Foundations of Software Engineering.

[16] Crosscutting Concerns in J2EE Applications. Mesbah, A. and Deursen, A. V. 2005. International Workshop on Web Site Evolution.

[17] Chong, S., Liu, J., Myers, A. C., Qi, X., Vikram, K., Zheng, L., Zheng, X.: Secure web application via automatic partitioning. In ACM Symposium on Operating Systems Principles, 2007.

[18] Web Modeling Language (WebML): a modeling language for designing Web sites. Ceri, S., Fraternali, P. and Bongio, A. 2000, Computer Networks.