논문 14-39B-05-12
The Journal of Korea Information and Communications Society '14-05 Vol.39B No.05
http://dx.doi.org/10.7840/kics.2014.39B.5.329

소프트웨어 모듈 재사용성을 고려한 JavaScript 전송 가속화 방안

김 기 정◆, 이 성 원°

A Scheme for Acceleration of JavaScript Transmission Considering Software Module Reusability

Gijeong Kim◆, Sungwon Lee°

요 약

상호교환적인 웹 서비스와 동적인 컨텐츠를 제공하기 위해서 JavaScript가 폭넓게 사용되고 있다. 하지만 웹 페이지를 구성하는 JavaScript의 크기와 개수가 점점 증가하고 있으며, 이는 웹 성능 저하로 이어지고 있다. 본 논문에서는 JavaScript가 프로그램이라는 점을 착안하여, 재사용성을 고려한 JavaScript 전송 가속화 방안을 제안한다. 네트워크 시뮬레이션을 통해서 성능 평가 및 분석을 수행하였으며, 제안 방안이 Gzip을 이용한 전송 방안보다 페이지 로딩 시간과 네트워크에 발생하는 트래픽 측면에서 향상된 성능을 제공함을 확인하였다.

Key Words : Web, JavaScript, Reusability

ABSTRACT

JavaScript is widely used to offer interactive web service and dynamic content. However, the number and size of JavaScript constituting the web page has been increasing steadily, and this circumstance leads to falling web performance. In this paper, we suggest a scheme for acceleration of JavaScript transmission considering software module reusability. We perform network simulation for the performance evaluation and analysis about the suggested scheme, and then confirm that the suggested scheme offers better performance in term of page loading time and the amount of traffic generated in the network than the transmission method using Gzip.

Ⅰ. 서 론

JavaScript는 서버와 클라이언트 간에 상호교환적인 웹 서비스를 제공하고, 시간이나 환경에 따라서 동적인 컨텐츠를 생성하는 것을 가능하게 한다. 하지만 JavaScript를 포함하는 웹 페이지를 수신할 경우, 해당 JavaScript를 수신하고 실행이 완료될 때까지 JavaScript 이후에 위치한 리소스에 대한 요청을 수행하지 않는 블록킹 현상이 발생시킨다. 웹 브라우저는 블록킹 현상을 발생시킴으로써 JavaScript를 안전하게 실행시킬 수 있는 독립적인 환경을 제공하지만, JavaScript를 수신하고 실행하는 동안 병렬 요청 및 수신을 할 수 없기 때문에 웹 성능을 저하시킨다[1,2].

본 논문에서는 JavaScript가 컨텐츠가 아닌 프로그램 코드라는 점을 착안하여, JavaScript 코드의 재사용성을 고려한 JavaScript의 전송 가속화 방안을 제안하고, 네트워크 시뮬레이션을 통해서 성능 평가 및 분석을 수행한다.

Ⅱ. 관련 연구

JavaScript의 전송 시간을 감소시키는 방안으로는 크게 Gzip을 이용해서 JavaScript 코드를 압축하여 전송하는 방안과 Google의 Closure Compile를 이용하여 코드를 최적화하는 방안이 있다.

Gzip을 이용해 JavaScript 코드를 압축하여 전송하는 경우, 전송되는 JavaScript의 크기를 압축함으로써 JavaScript의 전송 시간을 단축시킬 수 있다. 하지만 소스 코드의 크기를 줄이지 못하고, 인코딩과 디코딩 과정을 수행해야 하는 단점이 있다[3].

Closure Compile는 JavaScript 코드의 변수의 길이를 줄이고, 공백 문자와 주석과 같은 불필요한 코드를 제거함으로써 JavaScript 소스 코드의 크기를 줄이기 때문에 JavaScript의 전송 시간을 단축시킨다. 하지만

※ 본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학 IT 연구센터 육성지원 사업의 연구결과로 수행되었음 (NIPA-2014(H0301-14-1020))
◆ First Author : Department of Computer Engineering, Kyunghee University, kimgijeong@khu.ac.kr, 학생회원
° Corresponding Author : Department of Computer Engineering, Kyunghee University, drsungwon@khu.ac.kr, 종신회원
논문번호 : KICS2014-01-025, Received January 30, 2014; Revised April 9, 2014; Accepted May 7, 2014

Closure Compile는 프로그램의 재사용성을 고려한 효율적인 전송을 제공하지는 못한다[4].

Ⅲ. Core JavaScript 전송 방안 제안

Core JavaScript 전송 방안은 자주 사용되거나 자주 사용될 것이라고 예측되는 JavaScript 함수들을 선정하여 미리 전송하고, 서버가 클라이언트에게 JavaScript를 전송할 때 미리 전송된 함수들을 제외한 나머지 코드 부분을 전송하는 방안이다. 이 방안은 Core JavaScript Prefetching 과정과 인덱스 파일 및 JavaScript 전송 과정으로 구성된다.

그림 1은 Core JavaScript Prefetching 과정을 나타낸 그림이다. 서버는 자주 사용될 것이라고 예상되는 JavaScript 함수를 Core JavaScript로 미리 선정한다. 100 개의 함수가 Core JavaScript로 선정되었다고 가정하고, 서버는 Core JavaScript를 Gzip으로 압축하여 Core.gz 파일을 생성한다. 클라이언트는 전원이 OFF 또는 SLEEP인 상태에서 ON으로 전환되면 Core.gz 파일을 서버에게 요청하여 수신 받는다. 수신이 완료되면 Gzip을 이용해 압축을 해제하여 Core JavaScript를 획득한다.

그림 2는 인덱스 파일 및 JavaScript 전송 과정을 나타낸 그림으로, Core JavaScript Prefetching 과정이 완료된 이후에 수행된다. 웹 페이지는 2개의 JavaScript로 구성되어 있고, 각 JavaScript 코드는 100 개의 함수로 구성되어 있으며, Original_A.js는 70 개의 함수가 Core JavaScript의 함수로, Original_B.js는 80 개의 함수가 Core JavaScript의 함수로 구성된다.클라이언트는 웹 페이지인 index.html을 요청하고, 서버로부터 index.html을 수신한다. 그 다음 클라이언트는 웹 페이지를 구성하는 JavaScript를 요청하기 전에 Core.index 파일을 요청한다. 요청을 받은 서버는 각 JavaScript에서 Core JavaScript의 어떤 함수가 어떤 함수

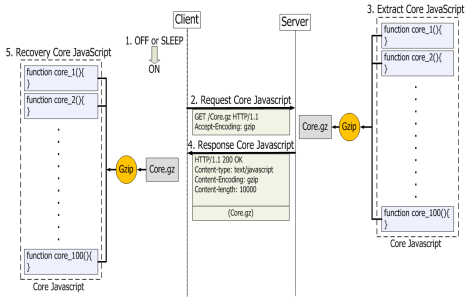

그림 1. Core JavaScript Prefetching 동작 과정
Fig. 1. Core JavaScript Prefetching process

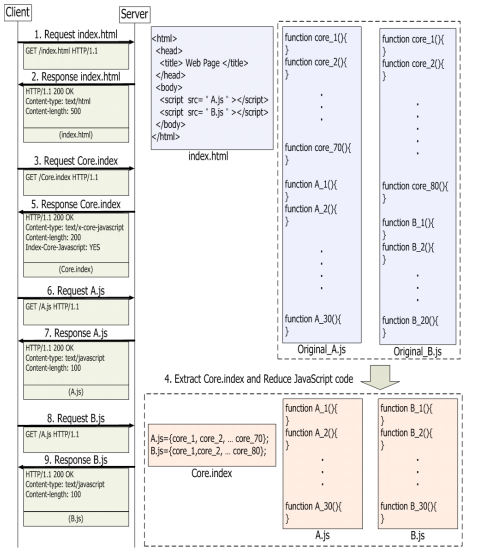

그림 2. Core JavaScript 전송 방안에서의 JavaScript 전송 과정
Fig. 2. JavaScript transmission process in Core JavaScript transmission scheme

가 사용되었는지를 나타내는 인덱스 파일인 Core.index 파일을 생성하여 클라이언트에게 전송한다. Core.index 파일을 수신한 클라이언트는 웹 페이지를 구성하는 리소스인 A.js와 B.js를 차례대로

요청한다. 요청을 받은 서버는 Origin_A.js와 Origin_B.js에서 Core JavaScript의 함수를 제거하여 A.js와 B.js를 생성한다. 서버는 A.js와 B.js를 클라이언트에게 전송함으로써 인덱스 파일 및 JavaScript 전송 과정을 종료하게 된다.

Ⅳ. 성능 평가 및 분석

성능 평가 및 분석을 하기 위해서 SMPL 라이브러리를 이용한 네트워크 시뮬레이션을 수행하였다. 시뮬레이션 시나리오는 클라이언트가 다수의 JavaScript를 포함하는 웹 페이지를 서버에게 요청하여 수신하는 것으로 한다. 2013년 2월 기준으로 전세계 4G 모바일 네트워크의 실제 평균 대역폭과 지연시간을 고려하여, 10 Mbps의 대역폭과 90 ms의 지연시간을 시뮬레이션에 적용하였다[5]. 웹 페이지의 크기 및 리소스의 개수는 Naver, Daum 등의 메인 웹 페이지를 분석하여 적용했으며, 메인 웹 페이지의 크기는 2000 Byte, 웹 페이지를 구성하는 리소스의 개수는 30개로 설정하였다. 웹 페이지를 구성하는 JavaScript의 크기는 html로 작성되는 메인 웹 페이지보다 훨씬 크며, 시간이 지남에 따라 JavaScript 크기의 증가폭이 점점 커지고

있음을 고려하여, 평균 1 MB로 지수분포를 가진다[2]. 캐시 적중률은 이산균등분포를 적용했다. JavaScript는 사용자 프로파일을 활용하여 동적인 데이터 생성과 GUI 생성을 제공하는데 주로 사용되며, 객체지향 프로그래밍 기법으로 주로 작성되고 있다. 객체지향 프로그래밍 기법의 평균 재사용률인 평균 70%를 JavaScript의 재사용률로 설정했다[6]. Gzip의 압축률은 파일의 내용과 크기에 따라 달라지며, 파일의 내용과 크기에 무관하게 평균으로 기대할 수 있는 Gzip의 압축 효율로 30%를 적용했다[7].

그림 3, 4, 5는 캐시 적중률에 따른 페이지 로딩 시간, 네트워크에 발생하는 메시지의 개수, 네트워크에 발생하는 트래픽을 나타낸 그래프이다. 페이지 로딩 시간과 네트워크에 발생하는 트래픽 측면에서, Gzip 전송 방안보다 Core JavaScript 전송 방안이 더 우수한 성능을 보인다. 또 캐시 적중률이 낮을수록 Core JavaScript 전송 방안의 성능이 극대화된다. 그 이유는 JavaScript 코드가 변경될 경우, Core JavaScript의 함수를 제외한 코드 부분과 인덱스 파일을 전송을 하는데, 인덱스 파일의 크기만큼 전송되는 데이터가 증가하지만 변경된 JavaScript만을 전송하는 과정에서 데이터를 감소시키는 이득이 훨씬 크기 때문이다. 하지만 네트워크에 발생하는 메시지의 개수 측면에는,

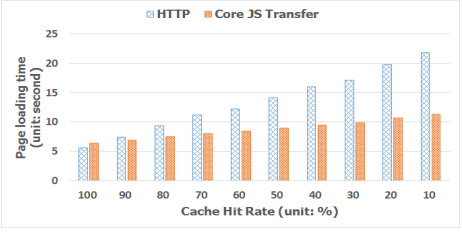

그림 3. Gzip 전송 방안과 Core JavaScript 전송 방안의 페이지 로딩 시간 비교 그래프
Fig. 3. A comparison graph of the page loading time in Gzip transmission scheme and Core JavaScript transmission scheme

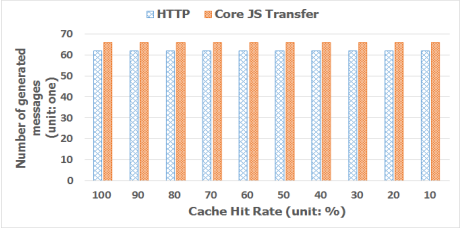

그림 4. Gzip 전송 방안과 Core JavaScript 전송 방안에서 발생하는 메시지의 개수 비교 그래프
Fig. 4. A comparison graph of the number of messages generated in Gzip transmission scheme and Core JavaScript transmission scheme

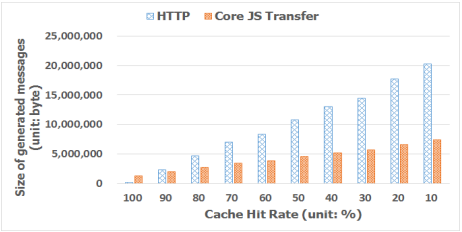

그림 5. Gzip 전송 방안과 Core JavaScript 전송 방안에서 발생하는 트래픽 비교 그래프
Fig. 5. A comparison graph of the amount of messages in Gzip transmission scheme and Core JavaScript transmission scheme

Gzip 전송 방안보다 Core JavaScript 전송 방안이 많은 메시지를 발생시킨다. 그 이유는 Core JavaScript Prefetching 과정과 인덱스 파일 전송 과정을 추가적으로 수행하기 때문이다. Core JavaScript 전송 방안은 Gzip 전송 방안 대비 페이지 로딩 시간을 평균 33.1% 향상시키고, 네트워크에 발생하는 트래픽을 평균 50.74% 감소시킨다.

V. 결 론

본 논문에서는 JavaScript의 재사용성을 고려한 전송 가속화 방안을 제안하고, 네트워크 시뮬레이션을 통해서 페이지 로딩 시간과 네트워크에 발생하는 트래픽 측면에서 성능이 우수함을 보였다. 제안 방안은 네트워크 장비의 프로토콜을 수정하거나 새로운 장비를 도입하지 않고, 서버와 클라이언트 간에 프로토콜의 수정만으로 웹 성능을 향상시킬 수 있기 때문에, 낮은 비용으로 웹 성능을 극대화할 수 있을 것으로 기대된다.

References

[1] Steve Souders, *Even Faster Web Sites*, O'Reilly, 2009.
[2] Mi-jung Choi et al, "Classification of Client-side Application-level HTTP Traffic," *KICS*, vol. 36, no. 11, pp. 1277-1284, Nov., 2011.
[3] Arvind Jain, Jason Glasgow, *Use Compression to make the web faster*(2012), Retrieved Jan., 10, 2014, from http://developers.google.com/speed/articles/use-compression/.
[4] Google, *Closure Tools*(2012), Retrieved Jan., 15, 2014, from http://developers.google.com/closure/.
[5] OpenSignal, *The State of US LTE*(2014), Retrieved Apr., 6, 2014, from http://opensignal.com/reports/state-of-lte/.
[6] P. Gandhi and P. K. Bhatia, "Estimation of generic reusability for object-oriented software an empirical approach," *ACM SIGSOFT Software Eng. Notes*, vol. 36, no. 3, pp. 1-4, May 2011.
[7] J.-L. Gailly, *GNU Gzip*(2013), Retrieved Apr., 8, 2014, from http://www.gnu.org/software/gzip/manual/gzip.html.