효율적인 웹 데이터 로딩을 위한 웹 캐싱 기법 분석

김현국 · 박진태 · 문일영

한국기술교육대학교

Analysis of Web Caching Techniques for Efficient Web Data Loading

Hyun-Gook Kim · Jin-Tae Park · Il-Young Moon

KOREATECH

E-mail : hy1392@koreatech.ac.kr

요 약

4차 산업혁명이 본격적으로 대두되기 시작하면서, 네트워크를 통하여 기기 간에 공유되는 데이터의 질과 양이 기하급수적으로 증가하기 시작하였다. 또한 이러한 데이터를 사람들이 접하는 기기의 범주도 데스크 탑에서 모바일 기기인 스마트폰, 스마트 패드, 스마트 워치 등 다양화되고, 소형화되기 시작하였다. 그 결과 데이터를 접하는 매개체가 변화하기 시작했고, 현대 사회인들이 가장 많은 양의 데이터를 접하는 곳은 스마트 기기라고 할 수 있다. 하지만 스마트 기기는 여전히 많은 양의 데이터를 한 번에 처리하기에는 네트워크 속도나 하드웨어 스펙 면에서 많은 아쉬움을 남긴다. 따라서 본 논문에서는 모바일 기기에서 보다 효율적으로 대량의 웹 콘텐츠들을 불러올 수 있는 캐시 기법인 Service Worker 활용, IndexedDB, LocalStroage에 대한 분석을 진행하고, 개선 방안을 제시하고자 한다.

ABSTRACT

As the fourth industrial revolution began to take off in earnest, the quality and quantity of data shared among devices began to increase exponentially over the network. In addition, the category of devices where people are exposed to such data has become diversified into desktops, mobile phones, smart phones, smart pads, and smart watches. As a result, the medium that contacts the data has begun to change, and the place where modern society has the most data is smart device. However, smart devices still leave much to be desired in terms of network speed and hardware specifications to handle large amounts of data at one time. Therefore, in this paper, we analyze the use of Service Worker, IndexedDB, and LocalStroage, which can retrieve a large amount of web contents more efficiently from mobile devices, and suggest ways to improve it.

키워드

Progressive Web App, IndexedDB, Localestorage, WebAssembly, JavaScript

Ⅰ. 서 론

4차 산업혁명이 본격적으로 대두되기 시작하면서, 네트워크를 통하여 기기 간에 공유되는 데이터의 종류가 이전에는 단순 텍스트, 이미지에 불과했던 데이터들이 변화하기 시작하였다. 먼저 공유되는 데이터의 질과 양이 이전에 비해 발전하고, 증가하였다. 데이터들은 이제 단순 텍스트나 이미지를 넘어서 영상, 게임 오브젝트, 3D 모델링 그리고 더 나아가 네이티브 코드까지 웹 어셈블리라는 형태를 통하여 웹을 통하여 전달되기 시작하였다 [1]. 또한 공유되는 데이터를 사람들이 접하는 기기의 범주가 변화하였다. 과거의 경우 대부분의 사람들이 데이터 혹은 정보를 접하는 매체가 TV나 라디오, 신문 등으로 수동적인 매체에 한정되었다. 하지만 요즘 사람들이 데이터를 접하는 매체는 TV, 신문, 데스크 탑에서 모바일 기기인 스마트폰, 스마트 패드, 스마트 워치로 점차 다양화되고, 소형화되기 시작하였다. 그 결과 데이터를 접하는 매개체의 변화에 따라 가장 많은 데이터가 공유되는 매체는 이제 스마트 기기가 되었고, 다양한 사람들이 스마트 기기 내에서 효율적으로 데이터를 제공하기 위해 노력하고 있다 [2]. 하지만 스마트 기기는 여전히 많은 양의 데이터를 한 번에 처리하기에는 여러 가지 제약사항들이 남아있다. 먼저 네트워크 속도이다. 일반적인 데스크 탑과 달리 유선이 아닌 무선으로 네트워크에 연결되기 때문에 통신 환경에 많은

영향을 받게 된다. 다음은 하드웨어 스펙이다. 일반적인 스마트 기기의 경우 소형화의 영향으로 기기가 포함하고 있는 CPU나 GPU, RAM의 성능이 일반적인 데스크 탑에 미치지 못한다. 지속적으로 스마트 기기의 성능이 발전하여 사람들이 불편함의 거의 느끼지 못하고 사용할 수 있도록 변화하였지만, 여전히 스마트 기기는 데이터를 처리하는 용도라고 보기 보다는 데이터를 시청하는 용도라고 보는 것이 맞을 것이다. 즉, 앞으로 웹이 해결해야할 가장 원대한 목표는 '스마트 기기에서 어떻게 하면 더 많은 양의 데이터를, 더 효율적으로 보여줄 수 있을까?' 일 것이다 [3].

하지만 실질적인 하드웨어 측면의 개선은 현 시점에서는 문제 해결을 위해 접근할 해법으로는 어울리지 않는다. 스마트 기기마다 파편화된 하드웨어 구조를 가지고 있으며, 그 형태 또한 상이하기 때문이다. 따라서 본 논문에서는 다른 한 측면인 스마트 기기의 네트워크 성능 측면에 초점을 맞추고자 한다.

II. 본 론

1. 웹 캐싱 기법 분석

스마트 기기에서 데이터를 접하는 경우, 가장 많이 사용되는 수단은 웹이다. 다양한 브라우저를 통하여, 각기 다른 사람들이, 다양한 데이터를 접하고 공유하게 된다. 하지만 웹 기술의 경우 서론에서 서술한 바와 같이 네트워크의 상태에 많은 영향을 받게 된다. 따라서 웹에서 사용자들이 데이터를 취득할 때, 사용자들이 좀 더 빠르고, 효율적으로 활용이 가능한 방법을 고안해내기 위하여 다양한 시도가 있었다. 그 중 가장 활발하게 연구가 진행되고 있는 분야 중 하나는 데이터 캐싱이다. 사용자의 기기 내에 데이터의 변동이 적어 지속적으로 네트워크를 통해 데이터의 갱신이 이루어질 필요가 적은 CSS, JavaScript와 같은 파일들을 미리 일정한 공간에 저장해 두는 것이다. 이후 사용자가 동일한 데이터에 대하여 요청을 할 경우, 미리 캐싱 된 데이터의 경우 네트워크를 통해 다시 받아오는 것이 아니라 저장된 데이터를 바로 불러와 속도를 개선시킨다. 따라서 본론에서는 사용자의 기기 내에 데이터를 저장하는 웹 캐싱 기법에 대하여 분석을 진행하였다.

	Chrome	Firefox	IE	Opera	Safari
IndexedDB	23	10	10	15	7.1
LocalStroage	4	3.5	8	10.50	4
Service Worker	40	33	X	24	X

그림 1. 웹브라우저 버전별 지원 형황

웹 페이지에 사용되는 데이터를 캐싱하는 기법은 크게 IndexedDB, LocalStorage, Service Worker 로 나눌 수 있다. 각각의 기술들이 지원하는 브라우저의 버전의 경우 그림 1과 같으며, Service Worker의 경우 지속적으로 지원 브라우저를 넓혀가 현재 모든 현대 브라우저 내에서 구동이 가능하다. 각 기술에 대한 자세한 설명은 아래와 같다.

1-1. IndexedDB

IndexedDB는 NoSQL기반의 대규모 데이터 저장 시스템이다. 파일이나 오브젝트와 같이 구조화 된 데이터를 클라이언트 측 저장소에 저장하기 위해 low-level API 형태로 제공되고 있다. IndexedDB 색인을 바탕으로 데이터를 저장하기 때문에, 저장된 데이터를 검색할 때 색인을 기반으로 한 고성능 검색이 가능하다. 또한 LocalStorage가 소량의 데이터를 저장하는 데 유용한 반면에 IndexedDB는 많은 양의 구조화 된 데이터를 저장하는 데 유용하다[4].

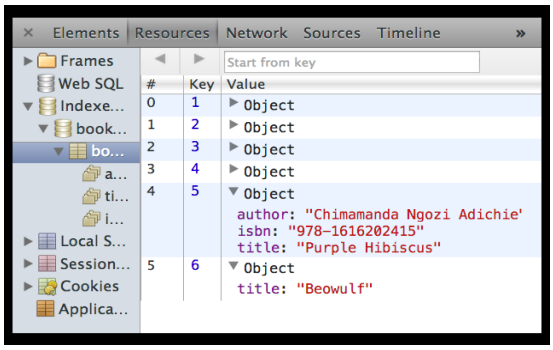

그림 2. 웹에서의 IndexedDB 예시

1-2. LocalStorage

LocalStorage는 사용자의 로컬로 환경에 데이터를 저장하여, 웹 어플리케이션에서 사용할 수 있도록 만드는 기법이다. HTML5 이전에는 모든 데이터를 쿠키에 저장하여 사용하거나 지속적은 http통신을 통해 서버로부터 받아서 사용해야 했다. 하지만 쿠키의 경우 보안 측면에서 취약한 부분이 많고, http통신의 경우 지속전이 통신으로 인한 오버헤드 발생으로 웹 어플리케이션의 성능을 저하시키는 부작용을 낳았다. 그에 비해 LocalStorage는 보다 안전하며 웹 사이트 성능에 영향을 미치지 않고 비교적 많은 양의 데이터를 로컬에 저장하고, 활용할 수 있도록 한다. 쿠키와 달리 저장 용량 한도도 최소 5MB로 크며, 로컬 환경 내에만 저장되기 때문에 외부 서버와 통신

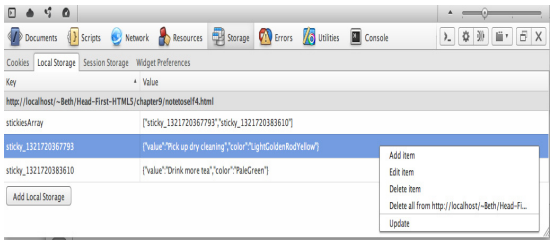

그림 3. 웹에서의 LocalStorage 예시

등으로 인한 오버헤드가 발생하지 않는다. 또한 LocalStorage는 도메인에 기반을 둔 서비스이기 때문에, 한 도메인 내에 포함된 모든 페이지는 동일한 데이터를 저장하고 액세스 할 수 있다 [5].

1-3. Service Worker

서비스 워커는 브라우저에 등록되어 백그라운드 단에서 독립적으로 실행되는 스크립트이다. 웹 페이지와는 별개로 동작하며, 웹 페이지 내에서 사용자의 입력에 반응해야 하는 요소들이 아닌 고정된 요소에 대한 기능을 지원한다. 현재 푸시 알림, 백그라운드 동기화, 캐싱과 같은 기능을 제공하고 있으며, 향후 주기적 동기화, 지오펜싱과 같은 기능 또한 지원할 예정이다. 또한 데이터를 indexedDB에 저장하고, 오프라인 상태 혹은 필요에 의해서 해당 데이터를 불러서 사용할 수도 있다 [6].

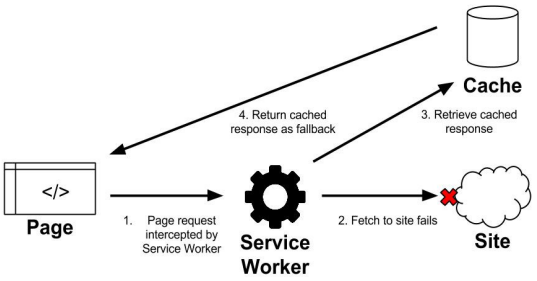

그림 4. Service Worker 동작 예시

2. 웹 성능 개선을 위한 기존 연구 분석

웹 어셈블리는 웹 브라우저 내에서 네이티브 언어로 작성된 코드를 2진 바이너리 코드로 변환하여 자바스크립트를 이용해 웹에서 동작하는 웹 기법이다. 이 때 웹에서 구동하고자 하는 것은 단순 데스크 탑의 응용 프로그램 뿐만 아니라 온라인 게임과 같은 용량이 거대한 프로그램 또한 포함되어 있다. 궁극적인 목표는 네이티브 코드로 된 프로그램을 웹에서 네이티브 환경과 흡사한 속도로 사용자들에게 제공하는 것이며, 2017년도 W3C 워킹그룹이 결성된 이후 이 분야에 대한 연구가 빠른 속도로 이루어지고 있다. 웹 어셈블리 모듈인 wasm파일의 경우에도 제공하고자 하는 컨텐츠가 게임, 3D모델링 등 대규모 리소스를 요구하는 경우, 파일의 크기가 비대해져 네트워크를 통해 사용자에게 제공되는 데에 있어 시간이 오래 걸리게 된다는 문제점이 발생한다. 따라서 웹 어셈블리가 이러한 문제를 해결하고, 데이터 제공 속도를 높이기 위하여 사용한 방법은 IndexedDB를 활용하는 방법이다. JavaScript에서 제공하는 API를 이용하여 컴파일 된 웹 어셈블리 파일인 wasm을 IndexedDB내에 저장하고, 사용자가 재요청 시 이를 활용하는 방법을 제시하였다 [1, 7]. 이를 통하여 웹 페이지가 갱신될 때 마다 새롭게 데이터를 받아와야 하는 문제는 해결되었다.

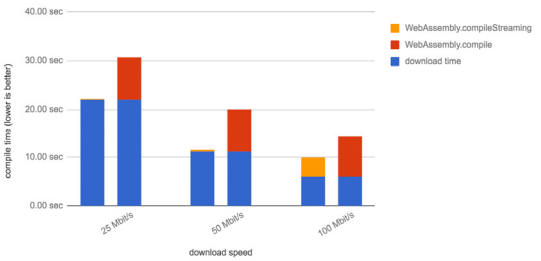

그림 5. Streaming Compile과 일반적인 Compile 방식의 소요 시간 비교

다른 연구로는 웹 어셈블리 모듈을 저장하는 과정이 아니라 다운로드하여 컴파일하는 과정에서의 개선을 제안하였다. 구글에서 2018년 4월에 게재한 “Loading WebAssembly modules efficiently”에서는 모듈을 네트워크를 통해 다운로드 받으면서 동시에 컴파일을 진행하는 Streaming Comile 방법을 제안하였다. 이를 통하여 그림 5에서 볼 수 있듯이 기존에 모든 모듈을 다운로드 받은 뒤 컴파일을 진행하는 방법에 비하여 컴파일 시간을 줄일 수 있었고, 이는 네트워크의 속도가 낮을수록 높은 효율을 나타내었다 [8],

하지만 안타깝게도 기존 연구에서 제안한 두 가지 방법에는 취약한 부분이 있다. 바로 코드가 변경되었을 경우 이를 반영하는 과정에 어려움이 있다는 것이다. 개발자에 의하여 버전이 변경되어 제공되는 모듈이 달라졌다거나, 내부 소스코드 내에 변화가 있을 경우 개발자가 직접 새로운 캐싱 코드를 작성하여 사용자들에게 제공하여야 한다. 또한 사용자가 직접 해당 페이지에 접속을 하여야만 코드가 갱신된 것을 알 수 있기 때문에 미리 갱신된 모듈들을 다운로드 받아 둔다던지, 백그라운드 내에서 다운로드를 진행하는 기능이 제공되지 않는다.

3. 웹 성능 개선을 위한 Service Worker 활용 방안 제시

따라서 본 논문에서 기존 캐싱 기법들과 연구들을 바탕으로 하여 개선점으로서 제시하고자 하는 방법은 Service Worker를 이용하는 방법이다. 먼저 데이터를 저장하는 방법의 경우 기존과 동일한 IndexedDB를 이용한다. 하지만 저장하는 방법에 있어 차이를 가진다. 사용자가 해당 웹 사이트에 접속하면 Service Worker를 사용자의 브라우저 내에 등록하게 된다. 이후 등록된 Service Worker가 개발자가 등록한 파일들을 자동으로 IndexedDB에 저장하고, 관리하게 된다. 이후 만약 개발자에 의하여 기존의 모듈에 변경이 발생하게 된다면, 등록된 Service Worker가 이를 감지하여 전체 모듈을 다시 다운로드 받는 것이 아니라 변경된 부분만을 선택적으로 갱신하여 소요되는 시간을 절약하게 된다. 또한 Service Worker는 백그라운드에서 동작이 가능하기 때문에 사용자가 해당 페이지에 데이터를 요청한 이후 다른 업

무를 진행 하더라도 데이터는 계속하여 다운로드 된다. 따라서 제안하는 방법을 적용하면 기존의 IndexedDB의 기술에 변경된 부분에 대한 선택적 갱신, 백그라운드 다운로드를 통하여 기존 연구에서 제안한 방법들의 한계로 지적되었던 문제점들을 해결할 수 있는 돌파구가 될 것이다.

III. 결 론

본 논문에서 제안하는 방법은 Service Worker를 이용하여 자동으로 웹 데이터 로딩에 필요한 요소들을 저장 및 백그라운드 로딩, 동기화 하는 방법이다. 이를 통해 기존의 연구에서는 다루지 않았던 사용자가 페이지를 보고 있지 않을 때 미리 Service Worker를 이용해 데이터를 저장하고, 사용자가 요청하면 결과 화면을 즉시 확인할 수 있도록 설계하였다. 따라서 다음 논문에서는 본 방식을 실제 웹 서비스에 접목하여 기존 논문에서 제안한 방법들과의 성능을 비교하고 실제 효용성에 대하여 논하고자 한다. 이와 더불어 Service Worker의 경우 지속적으로 새로운 기능들이 추가되고 있는 과도기적 시기이기 때문에, 앞으로 웹 데이터 로딩의 속도를 개선할 수 있는 더욱 다양한 아이디어들이 제안될 것이다.

감사의 글

본 연구는 중소벤처기업부와 한국산업기술진흥원의 지역특화산업육성(R&D) 기술개발사업으로 수행된 연구결과입니다.

참고문헌

[1] Haas, Andreas, et al. "Bringing the web up to speed with WebAssembly." Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, 2017.
[2] Watt, Conrad. "Mechanising and verifying the WebAssembly specification." Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. ACM, 2018.
[3] Finney, Richard, and Daoud Meerzaman. "Chromatic: WebAssembly-Based Cancer Genome Viewer." Cancer Informatics 17 (2018): 1176935118771972.
[4] https://developer.mozilla.org/ko/docs/Web/API/ServiceWorker
[5] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
[6] https://developers.google.com/web/fundamentals/primers/service-workers/?hl=ko
[7] http://www.clipcode.net/training/clipcode-webassembly-devguide.pdf
[8] https://developers.google.com/web/updates/2018/04/loading-wasm