JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

기반 기술

J. Adv. Navig. Technol. 23(2): 158-165, Apr. 2019

서비스워커와 해시를 이용한 통신 데이터 소모 감소를 위한 웹 콘텐츠 갱신 알고리즘 설계

Design of Web Content Update Algorithm to Reduce Communication Data Consumption using Service Worker and Hash

김 현 국 · 박 진 태 · 최 문 혁* · 문 일 영
한국기술교육대학교 컴퓨터공학부

Hyun-gook Kim · Jin-tae Park · Moon-Hyuk Choi* · Il-young Moon
Department of Computer Engineering, KOREATECH, Chungcheongnam-do, 31253, Korea

[요 약]

기존 웹 페이지는 사용자가 해당 페이지를 요청할 때 마다 새로운 페이지를 다운로드 받아 사용자에게 제공하였다. 따라서 동일한 페이지를 사용자가 반복해서 요청할 경우 동일한 리소스에 대한 다운로드만을 반복하게 된다. 이는 불필요한 데이터의 소비를 발생시키는 요인이다. 본 논문에서는 사용자와 서버간의 불필요한 요청에 의해 발생하는 데이터의 소모를 감소시키고, 콘텐츠 제공 속도를 향상시키는 대에 초점을 맞추었다. 따라서 이를 방지하기 위하여 본 논문에서는 캐싱 시스템과 사용자가 요청하는 파일의 변경을 감지할 수 있는 해시 함수를 이용한 해시 값의 비교를 통해 항상 최신 캐시를 유지하면서 데이터 소모를 줄일 수 있는 알고리즘에 대하여 논하고, 웹 콘텐츠의 속도 향상을 위한 서비스 워커 기반의 캐싱 시스템을 설계하고 성능을 평가하였다.

[Abstract]

The existing web page was downloaded and provided to the user every time the user requested the page. Therefore, if the same page is repeatedly requested by the user, only the download for the same resource is repeated. This is a factor that causes unnecessary consumption of data. We focus on reducing data consumption caused by unnecessary requests between users and servers, and improving content delivery speed. Therefore, in this paper, we propose a caching system and an algorithm that can reduce the data consumption while maintaining the latest cache by comparing the hash value using the hash function that can detect the change of the file requested by the user.

Key word : Web performance, Web cache, Service Worker, Progressive Web App.

https://doi.org/10.12673/jant.2019.23.2.158

Received 20 February 2019; **Revised** 5 April 2019
Accepted (Publication) 20 April 2019 (30 April 2019)

***Corresponding Author ; Moon-Hyuk Choi**

Tel: +82-10-6479-6754
E-mail: moon1225@koreatech.ac.kr

Ⅰ. 서 론

기존 웹 페이지는 사용자가 해당 페이지를 요청할 때 마다 새로운 페이지를 다운로드 받아 사용자에게 제공하였다. 따라서 동일한 페이지를 사용자가 반복해서 요청할 경우 동일한 리소스에 대한 다운로드만을 반복하게 된다. 이는 불필요한 데이터의 소비를 발생시키는 요인이다. 하지만 사용자들이 사용할 수 있는 네트워크는 제한되어있고, 항상 소비자들은 적은 네트워크 소모로 많은 정보를 획득하기를 원했다. 따라서 캐싱을 비롯한 다양한 기술들이 사용자들을 만족시키기 위하여 등장하였다 [1],[2].

하지만 캐싱에는 치명적이 약점이 있다. 캐싱은 사용자가 해당 페이지를 처음 접근하였을 당시의 정보를 그대로 브라우저 내에 저장하기 때문에 캐시가 삭제되기 전 까지 항상 동일한 결과를 사용자에게 제공하게 된다. 따라서 사용자가 페이지를 방문한 뒤 해당 페이지의 css요소가 변경되었다 할지라도 사용자는 이를 인지하지 못하고 이전에 브라우저 내에 캐싱된 정보만을 받아보게 된다. 이에 따라 사용자가 항상 최신 정보를 제공받고 있다는 것을 확신할 수 없다. 이는 서비스를 제공하는 제공자 입장에서 치명적인 요소로 작용한다. 실제 서비스의 사용자가 잘못된 정보를 받게 되기 때문이다 [3],[4].

따라서 본 논문에서는 기존의 캐싱 시스템이 가지는 이점을 활용하되 이를 개선하기 위하여 서버 파일의 해시 값을 이용해 파일의 최신 상태를 검증하고, 서비스워커를 이용해 사용자의 브라우저와 별개로 백그라운드 내에서 캐싱된 파일들의 최신화를 진행하여 네트워크 리소스의 소모를 낮추고, 사용자의 행동과 별개로 최신화 작업을 진행하여 서비스의 질을 높일 수 있는 알고리즘에 대하여 논하고자 한다.

Ⅱ. 관련 기술 조사

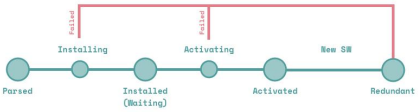

그림 1. 서비스워커 생명주기
Fig. 1. 서비스 워커 Lifecycle.

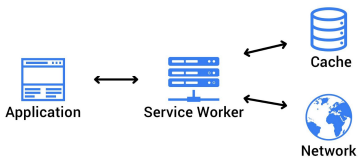

그림 2. 서비스 워커와 cache api를 이용한 서비스 예시
Fig. 2. Example service using 서비스 워커 and cache api.

본 절에서 실제 논문에서 제안한 시스템에 사용된 핵심 기술에 대하여 분석을 진행하고자 한다. 분석을 진행하고자 하는 기술은 브라우저와 별개로 동작하여 사용자의 요청, push 이벤트, 오프라인 환경 등을 처리할 서비스 워커 그리고 사용자가 요청한 페이지를 브라우저 내에 저장하여 관리하고, 서버로부터 동일한 파일에 대한 재 다운로드 과정 없이 캐싱된 파일을 즉시 제공하기 위한 cache api 이다. 각 기술은 현재 모든 modern browser에서 지원하고 있다.

2-1 서비스 워커

서비스 워커는 브라우저가 백그라운드에서 실행하는 스크립트이다. 웹 페이지와 별개로 작동하며, 웹 페이지 혹은 사용자의 상호작용이 필요하지 않은 기능들에 대해 확장성을 제공하고 있다. 기존의 웹이 온라인 환경에서 서비스를 제공하는 것에 주목한 반면 서비스 워커는 오프라인 서비스의 측면도 고려하고 있기 때문에 개발자가 제공할 수 있는 서비스의 폭이 넓다. 서비스 워커 이전에도 오프라인 서비스를 지원하기 위하여 appcahce라는 api가 존재하였지만 현재 이는 single page app에서는 정상 작동 하지만 여러 페이지로 구성된 일반적인 사이트에서 문제점이 많아 이를 해결하기 위해 제안되었다. 서비스워커는 프로그래밍 가능한 네트워크 프록시로 페이지의 네트워크 요청 처리 방법을 제어할 수 있다. 따라서 자바스크립트로 작성되지만 웹 페이지의 dom에는 직접 접근할 수 없으며 필요한 경우 postmessage 인터페이스를 이용하여 웹 페이지로부터 전달된 메시지에 대한 응답을 처리하는 방식으로 dom에 접근하게 된다. 브라우저 상태와 별개로 서비스 워커는 사용하지 않을 때 종료되고 다음에 필용할 때 다시 시작되어 서비스를 제공한다 [5].

서비스 워커의 생명주기는 웹 페이지와 완전히 별개로서 동작한다. 먼저 서비스 워커를 사이트에 설치하기 위해서는 자바스크립트 코드를 이용해야 한다. 네트워크를 통해 다운로드 받은 서비스 워커는 브라우저의 백그라운드 단에서 설치를 진행하고 서비스를 시작한다. 설치 과정에서는 주로 사용자들에게 제공되기 위한 정적인 파일들에 대한 캐싱 과정이 함께 수행된다. 설치가 실패할 경우 사용자가 해당 페이지를 재방문 하였을 때 다시 설치를 시도한다. 설치가 완료되면 사용자에게 서비스를 제공하기 위한 활성화 단계가 진행되고 이 단계에서 주로 오래된 캐시를 관리한다. 활성화 단계를 거친 서비스 워커는 등록된 서비스 범위 내에서 발생하는 모든 페이지에 대하여 네트워크 요청을 관리하게 된다. 서비스 워커에 제어 권한이 부여된 경우 서비스 워커는 메모리 절약을 위해 종료되거나, 페이지의 네트워크 요청이나 메시지에 대하여 페치 및 message 이벤트를 처리한다 [6].

다음 그림 1은 실제 서비스 워커의 생명 주기를 간단한 그림으로 나타낸 것이다. 서비스 워커의 생명 주기는 크게 다운로드, 파싱 및 실행, 설치 ,활성화, 업데이트 순으로 이루어진다.

사용자의 브라우저 내에서 .register()가 호출되면 서비스 워커의 다운로드가 진행된다. 만약 이 과정에서 다운로드에 실패하거나 실행 과정에서 오류가 발생한 경우 설치가 거부되고 다운로드 된 서비스 워커가 삭제된다. 서비스 워커가 다운로드 된 이후 처음으로 실행되는 install 이벤트는 서비스 워커가 실행되는 즉시 발생되고, 서비스 워커별로 한 번만 호출된다. 이 과정에서 사용자에게 제공할 정적 파일에 대한 캐싱을 진행할 수 있다. 서비스 워커가 클라이언트를 제어하고 push, sync, 페치 등의 이벤트를 처리할 준비가 되면 activate 이벤트가 발생한다. 이후 24시간 내에 업데이트 확인이 없는 상태에서 페치와 같은 이벤트가 발생하거나 서비스 워커의 url이 수정된 경우 update 이벤트가 발생한다. update는 기존의 서비스 워커의 동작과 동시에 진행되며 새로운 서비스 워커가 다운로드 되어 install 될 때 까지 기존의 서비스 워커가 역할을 수행한다. 새로운 서비스 워커가 성공적으로 설치되면 기존의 서비스 워커는 redundant 상태로 변경되고 새롭게 설치된 서비스 워커가 활성화 되어 서비스를 수행하게 된다 [7].

본 논문에서는 서비스 워커를 활용하여 사용자의 페치 이벤트 및 웹 페이지 이탈, 브라우저 종료에 대한 이벤트를 인지하고 해당 요청에 대한 응답 값을 제어하기 위하여 서비스 워커를 사용하였다.

2-2 cache api

cache api는 웹 페이지 네트워크 요청에서 발생하는 request/response 객체 쌍에 대한 값을 브라우저 내에 저장하기 위한 인터페이스이다. cache api 브라우저의 윈도우 Scope에도 노출되어 있기 때문에 자바스크립트 내에서 쉽게 접근하여 사용할 수 있다는 장점이 있다. 하나의 캐시 저장소에 다수의 이름이 지정된 cache 객체를 저장할 수 있으며, 개발자는 api를 통해 캐시의 저장, 삭제, 갱신을 관리하게 된다. 이미 저장되어 있는 캐시에 대하여 명시적인 요청이 발생하지 않으면 해당 캐시는 삭제되거나 만료되지 않기 때문에 생성된 캐시에 대해서는 유지관리에 대한 기능을 정의해야 한다. 만일 캐시에 대한 관리가 이루어지지 않을 경우 각 브라우저에 주어진 캐시 저장소의 용량이 한계에 임박하여 서비스 제공에 문제를 야기할 수 있다. 각 브라우저별로 캐시에 할당된 저장소 용량은 storageestimate api를 통해 확인할 수 있다. 캐시의 저장, 삭제 및 갱신은 모두 promise 기반의 비동기 처리로 이루어진다 [8], [9].

저장된 캐시에 대하여 지속적인 관리를 요하기 때문에 주로 개별적으로 사용되기 보다는 서비스 워커와 함께 사용된다. 서비스 워커의 install 이벤트 단계에서 보통 사용자에게 정적으로 제공되는 파일들에 대한 캐싱을 진행하고, activate 단계에서 이전에 저장한 오래된 캐시에 대해서 삭제를 수행한다. 이후 페치 혹은 사용자가 정의한 sync 이벤트에 의해 새로운 데이터의 캐싱 혹은 삭제가 이루어진다. 그림 2는 실제 등록된 브라우저에 적용된 서비스 워커가 cache api와 함께 동작하는 일반적인 과정을 간략하게 나타낸 것이다. 만약 사용자에 의해 어플리케이션에서 페치 이벤트가 발생할 경우 서비스 워커는 이를 감지하여 사용자가 요청한 파일이 캐시에 존재하는지 확인한다. 만약 해당 파일이 캐시 내에 존재할 경우 사용자에게 캐시 되어 있는 데이터를 즉시 반환한다. 만약 요청한 데이터가 캐시 내에 존재하지 않을 경우 비로소 서버에 해당 파일을 요청 후 다운로드하여 사용자에게 제공하게 된다 [10].

본 논문에서는 네트워크 요청을 통해 서버로부터 전달받은 파일의 데이터 및 해시 값을 저장하고, 이를 기반으로 캐싱된 파일과 서버의 파일의 일치 여부를 판별하여 항상 최신 파일을 캐시에 저장해 두고 사용자에게 제공하기 위하여 cache api를 활용하였다. 3장에서는 실제 서비스 워커와 cache api를 바탕으로 하여 웹 콘텐츠의 속도 향상을 위한 서비스워커 기반의 캐싱 시스템을 설계하고자 한다.

Ⅲ. 서비스워커 및 해시 기반의 서비스 설계

본 절에서 제안하고자 하는 시스템은 다음 그림 3과 같은 구조를 가진다. 사용자들은 데스크톱, 랩톱, 스마트폰, 태블릿, 라즈베리파이 등 모던 브라우저가 사용 가능한 모든 기기에서 본 논문에서 제안하는 기술을 사용할 수 있다. 사용자가 원하는 기기를 통해 브라우저를 실행, 논문에서 제안된 시스템이 적용된 웹 페이지에 접근하게 될 경우 자동적으로 서버에 정의된 서비스워커가 사용자의 브라우저에 다운로드 된다. 이후

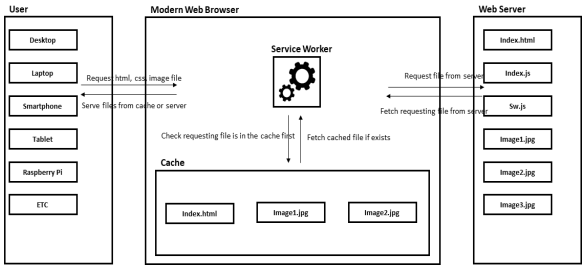

그림 3. 시스템 구조도
Fig. 3. System Architecture.

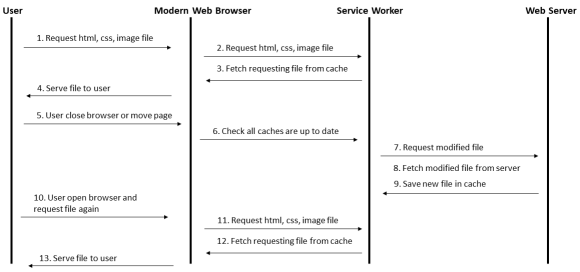

그림 4. 시스템 동작도 – 사용자가 요청한 파일이 캐시에 존재하는 경우
Fig. 4. System operation diagram - If the file requested by the user exists in the cache.

를 거치며 사용 가능한 상태로 전환된다. 서비스워커가 사용 서비스워커는 브라우저의 동작과 별개로 설치 및 활성화 단계 자의 브라우저에 성공적으로 등록된 이후에는 사용자가 발생 하는 모든 요청들은 직접적으로 서버로 바로 전송되는 것이 아니라 등록된 서비스워커를 통해 전송되게 된다. 이 과정에서 서비스워커는 사용자의 요청을 확인하여 사용자가 요청하고자 하는 파일이 만약 현재 브라우저의 캐시 내에 존재할 경우 네트워크를 통하여 파일을 다시 다운로드 받는 것이 아니라 캐싱 되어 있는 파일을 바로 전달하여 불필요한 다운로드에 소요되는 시간을 감소시키다. 만약 파일이 캐시 내에 존재하지 않는다면 서비스워커는 해당 요청을 서버로 전송하여 사용자가 요청한 파일을 다운로드 받고, 이를 먼저 캐시 내에 저장한다. 이후 캐시에 저장된 파일을 사용자에게 제공하여 이후에 또 다시 사용자가 파일을 요청할 경우 다운로드가 반복되지 않도록 한다. 본 논문에서 제안하는 서비스는 단순 캐싱뿐만 아니라 사용자가 항상 최신 파일에 대한 정보만을 받아올 수 있게 만들기 위하여 사용자의 페이지 이동 혹은 브라우저 종료를 감지하여 해당 상황이 발생할 경우 기존에 저장된 캐시들에 대하여 변경된 파일에 대한 캐시 갱신을 진행한다. 이는 사용자가 브라우저를 사용하고 있지 않더라도 네트워크가 연결되어 있는 상황이라면 별개로 동작하여 항상 최신 파일을 유지할 수 있도록 유도한다.

최종적으로 본 논문에서 제안하고자 하는 시스템은 상기 기능뿐만 아니라 사용자가 브라우저를 종료하거나, 다른 웹 페이지로 이동한 경우에도 이를 감지하고 데이터를 갱신하기 위한 이벤트를 서비스워커에 전송한다. 데이터 갱신을 위한 이벤트를 인지한 서비스 워커는 먼저 캐시에 저장된 해시 값과 서버로부터 다신 전송받은 해시 값이 동일한지 확인한다. 만일 해시 값에 변동이 있을 경우 캐시 내에 저장되어 있는 해당 데이터를 서버로부터 다시 다운로드 받아 저장하여 둔다. 해당 과정은 브라우저와 별개로 동작하게 된다. 따라서 사용자는 이러한 과정을 인지하지 못하지만, 논문에서 제안된 서비스가 적용된 웹 페이지를 방문할 때 마다 항상 최신 데이터로 갱신된 웹 페이지를 확인하게 된다. 자세한 동작은 다음 그림 4와 같다.

Ⅳ. 서비스워커 및 해시 기반의 서비스 구현 및 성능 평가

표 1. hash.php 소스 코드
Table 1. hash.php source code.

```
<?php
$file = $_GET["filename"];

echo json_encode(md5_file($file));
?>
```

표 2. index.html 소스 코드
Table 2. index.html source code.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initi
al-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <style>
    img {
      width: 500px;
      height: 500px;
      display: inline-block;
    }
  </style>
</head>

<body>
  <img src="./img-1.jpg" alt="">
  <img src="./img-2.jpg" alt="">
  <img src="./img-3.jpg" alt="">
</body>

<script src="./index.js"></script>
</html>
```

표 3. index.js 소스 코드
Table 3. index.js source code.

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker
    .register('./sw.js')
}

window.addEventListener("beforeunload", function (event) {
    event.preventDefault();
    navigator.serviceWorker.ready.then(function (reg) {
        return reg.sync.register('syncData').then(() => {
            console.log('client sync registered')
        })
    })
});
```

표 4. sw.js 소스 코드 중 캐시 최신화 함수 정의부
Table 4. sw.js Source code update cache function definition.

```
self.addEventListener('sync', event => {
  if (event.tag == 'syncData') {
    console.log("syncData")
    if (navigator.onLine){
      caches.open(PRECACHE).then(cache => {
        cache.keys().then(cachesToRefresh => {
          cachesToRefresh.forEach(toRefresh => {
            let target = toRefresh.url.split("/")[toRefresh.ur
l.split("/").length-1]
            if(target.length>0){
              let hash

              cache.match("/"+target)
                .then(function (response){
                  return response.json()
                }).then(function (data){
                  hash = data
                  return data;
                }).then(function(data){
                  fetch("hash.php?filename="+target, {
                  cache: "reload",
                  credentials: "omit"
                  })
                  .then(function (response) {
                    return response.json();

                  }).then(function(data){
                    let state = data == hash
                    console.log(`${data} : ${hash} ==>
${state}`)
                    return state;
                  }).then(function(state){
                    if(state == false && !toRefresh.url.in
cludes("sw-default")){
                      let url = toRefresh.url.replace(targ
et, "sw-default/"+target)
                      var fetchPromise = fetch(url, {
                        cache: "reload",
                        credentials: "omit"
                      }).then(function  (networkRespons
e) {
                        cache.delete(url).then(function()
{
                          cache.put(url, networkRespon
se.clone())
                        })
                      }).then(function(){
                        fetch("hash.php?filename="+tar
get, {
                        cache: "reload",
                        credentials: "omit"
                        })
                        .then(function (response) {
                          cache.delete("/"+target).then
(function(){
                          cache.put("/"+target,  respons
e.clone())
                          console.log(`refresh cache of
/${target}`)
                        })
                        })
                      })
                      console.log(`페치  ${toRefresh.ur
l} from server on sync`);
                    }
                  })
                })
            }
          });
        })
      })
    }
  }
})
```

표 5. 실험 측정 결과 – Fast 3G 네트워크 환경
Table 5. Experimental Results - Fast 3G Network Environment.

Case	1	2	3	4	5	6	7	8	9	10
1	59.2	59.7	58.3	58.9	57.1	59.3	57.9	58.9	58.6	58.8
2	57.9	1.29	1.19	1.26	1.24	1.22	1.22	1.31	1.28	1.27
3	58.2	1.25	1.26	1.29	1.30	1.21	1.28	1.18	1.22	1.29

표 6. 실험 측정 결과 – 100Mbps 네트워크
Table 6. Experimental Results - 100Mbps Network.

Case	1	2	3	4	5	6	7	8	9	10
1	1.20	1.26	1.29	1.31	1.29	1.24	1.25	1.25	1.24	1.31
2	1.26	1.13	1.22	1.25	1.21	1.23	1.25	1.21	1.22	1.29
3	1.28	1.25	1.23	1.29	1.18	1.22	1.28	1.25	1.23	1.26

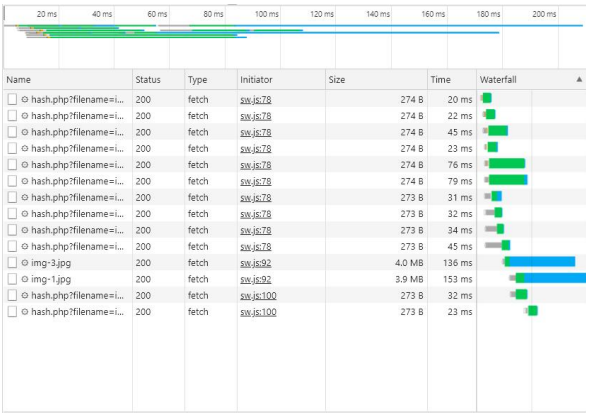

그림 5. 캐시 갱신시 발생하는 네트워크 트래픽
Fig. 5. Network traffic that occurs during cache updates.

본 절에서는 실제 제안한 시스템을 실제 코드를 작성하여 구현하고, 이후 진행 될 실험의 구조를 정의하고자 한다. 본 절에서 구현하고자 하는 소스코드는 크게 4부분으로 나뉘며 전체 구성은 다음 표의 내용과 같다.

다음 표1은 요청한 파일에 대한 해시 값을 반환하는 함수이다. 모든 요청은 get 방식의 인자 값으로 파일명을 의미하는 filename을 전달하며, 서버에서 이에 해당하는 파일을 찾아 md5 해시 함수 기반의 값을 산출하여 json 형태로 반환하고 있다.

다음은 표 2는 실제 index.html의 구현에 사용된 소스코드이다. 웹 표준인 html5를 기준으로 작성되었으며, 사용자에게 이미지를 제공하기 위하여 <img></img>태그를 다수 포함하고 있다. 본 논문에서는 실험을 위하여 이미지 파일을 사용하였는데 이는 이미지 파일의 경우 웹을 통해 사용자가 가장 자주 접하는 미디어 콘텐츠 중 하나이며 일반적인 텍스트와 달리 비교적 큰 용량을 가지고 있기 때문이다. 따라서 사용자에게 이미지가 제공될 때 다른 콘텐츠보다 오랜 시간이 소요되어 이미지를 이용하여 기존 시스템과의 속도 비교를 진행하여 보편적이 시스템에서도 본 논문에서 제안하는 시스템이 적용될 수 있음을 보여주고자 하였다. 각 이미지 파일의 용량은 약 4-5 mb 내외이며 가로, 세로(width, height)를 각 500 px로 고정하여 브라우저 상에서 사용자가 시각적으로 충분히 크게 인지할 수 있도록 하였다.

다음 표 3은 실제 index.js의 구현에 사용된 소스코드이다. 사용자가 웹 페이지에 접근하였을 때 사용자의 브라우저에 서비스워커를 등록하는 역할을 수행하고 있다. 현재 navigator.serviceWorker.register('./sw.js') 함수를 이용하여 sw.js라는 본 논문을 위해 작성된 서비스워커 파일을 서비스에 등록하고 있다. 추가적으로 본 논문에서 사용자가 제공되고 있는 페이지를 벗어나 다른 화면으로 이동, 브라우저를 종료 하였을 경우 사용자의 브라우저와 별개로 기존에 방문한 웹 페이지에 대한 캐시를 갱신하여 최신 데이터로 변경하는 작업을 수행하기 위해 beforebound 이벤트에 대해 서비스워커에 syncData라는 개별적으로 정의된 sync 이벤트를 등록하도록 하였다.

다음 표 4는 본 논문에서 제안하는 시스템의 핵심으로 사용자가 브라우저를 종료하거나 페이지를 이동했을 때 발생하는 sync 이벤트에 대한 처리 함수이다. 사용자가 브라우저 종료 혹은 페이지 이동을 하여 이벤트를 발생시킬 경우 syncData라는 이벤트가 서비스워커에 전달되며, 서비스워커는 이 이벤트를 인지한 뒤 현재 사용자의 네트워크 상태가 온라인 상태인지 navigator.onLine을 이용해 상태를 확인한다. 이후 사용자가 인터넷에 연결된 상태이라면 현재 캐시에 저장되어 있는 데이터들에 대해서 최신 캐시 정보가 있는지 확인하고, 이를 갱신하여 캐시에 저장하여 준다. 이를 통해 사용자가 항상 최신 캐시 정보를 받아볼 수 있도록 한다. 항상 서버로부터 최신 파일을 유지하기 위하여 install 과정에서 저장한 각 파일의 해시 값을 서버로부터 다시 받아와 모두 동일한지 비교한 후 해시 값이 변동된 파일에 대해서는 새로운 해시 값을 저장하고, 서버로부터 변경된 파일을 다운로드 받아 캐시 값을 갱신하게 된다. 모든 과정에서 해시 값을 기반으로 오직 변경된 파일만을 서버로부터 다시 다운로드 받아 갱신하고 있어 불필요한 네트워크 리소스 낭비를 줄이고 있다. 해시 값을 받아오는 http 요청에 다운로드 되는 데이터의 경우 약 200 byte로 데이터의 크기가 매우 작으며, 느린 네트워크 환경에서도 충분히 구동될 수 있다.

실험은 다음과 같은 환경을 기준으로 진행되었다. 1은 캐시 시스템이 적용되지 않은 일반적인 웹 사이트, 2는 구글에서 제공하는 기본 서비스워커 코드를 적용한 웹 사이트, 3은 본 논문에서 제안하는 시스템이 적용된 웹 사이트이다. 각 case에 대하여 10회 씩 실험을 진행 하였으며 페이지에 접속한 뒤 서버의 이미지를 임의의 대체 이미지로 변경하였다. 이후 사용자가 브라우저를 종료하고 다시 해당 페이지에 접속하였을 경우 모든 콘텐츠가 웹 브라우저에 모두 로딩 되어 보여지기 까지 소요되는 시간을 측정하였다.

실험 결과를 살펴보면 네트워크 속도가 빠른 환경에서는 큰 차이를 볼 수 없지만, 네트워크의 속도가 제한된 환경에서는 첫 방문 이후에 페이지를 다시 방문하였을 때 소요되는 시간에서 큰 차이를 보이는 것을 확인할 수 있다. 이는 논문에서 제안하고 있는 서비스가 사용자가 다시 통신을 요청하였을 때 기존에 미리 캐싱하고 있던 파일을 제공하여 다운로드 없이 콘텐츠를

조회할 수 있도록 서비스를 제공하고 있기 때문이다.

시간상으로 보았을 때 에는 2와 3의 성능에는 큰 차이가 없다는 것이 확인할 수 있었다. 하지만 2의 경우 실험 2에서 이미지가 변경된 것을 감지하지 못하고 계속해서 이전에 캐싱 해 둔 이미지를 불러와 사용자에게 제공하였다. 본 논문에서 제안한 3의 경우 사용자가 페이지를 이동하거나 브라우저를 종료할 경우 서비스워커에 데이터 갱신을 위한 이벤트를 등록하고 서버와의 파일 해시 값 검증으로 통해 데이터를 브라우저 상태와 별개로 백그라운드에서 갱신하는 과정을 거쳤다. 따라서 정상적으로 서버로부터 갱신된 파일을 다운로드 받아 사용자에게 제공할 수 있었다.

실제 2개의 이미지가 변경된 것에 대하여 캐시가 갱신되는 과정에서 나타난 네트워크 트래픽의 경우 다음 그림 5에서 확인할 수 있는 결과와 같다. 파일을 받아오는 페치 이벤트에 대한 근원이 sw.js로 서비스워커에 의해 발생한 이벤트임을 확인할 수 있다. 해당 과정에서 서버로부터 파일의 해시 값을 받아와 비교하기 위해 다수의 해시 값을 받아오는 페치 요청이 발생하였으며, 실질적으로 파일의 다운로드가 일어난 것은 서버의 파일이 변경된 “img-1.jpg”와 “img-3.jpg” 단 두 파일에 대해서만 일어난 것을 확인할 수 있다. 해시 값을 요청하는 페치 과정에서 발생한 트래픽 사이즈는 273 byte 이며, 각 이미지를 다운로드 받는데 발생한 트래픽은 4MB로 나타났다. 이를 통해 불필요한 파일에 대해서는 다운로드를 요청하는 페치 이벤트를 발생시키지 않아 전체 페이지를 구성하는 index.html, index.js, sw.js, im-1.jpg, img-2.jpg, img-3.jpg 총 6개의 파일 중 변경된 img-1.jpg, img-3.jpg만 다운로드하여 네트워크 리소스의 낭비를 줄인 것을 확인할 수 있었다.

Ⅴ. 결 론

본 논문에서는 제안한 알고리즘을 직접 구현하고 실험을 진행하여 캐시를 갱신하기 위하여 md5 해시 함수 기반의 해시 값 검증 알고리즘을 적용하여 전체 캐시된 파일을 갱신하는 것이 아니라 서버에서 파일의 변경이 일어난 파일에 대한 캐시 값만을 다시 다운로드 받도록 하였다. 해시 값을 검증하는 과정에서는 직접 파일을 요청하는 것이 아니라 서버에 작성된 파일에 대한 해시 값을 반환하는 api를 구축하여 사용하였으며, 이 과정에서 실제 파일이 아니라 단순 해시 값의 텍스트만을 다운로드 받아 통신 과정에서 발생하는 데이터의 소모를 줄였다. 실제 api를 통해 해시 값을 가져오는 요청에 대한 데이터 크기는 chrome 브라우저를 기반으로 측정하였을 때 약 273byte로 일반적인 html, css javascript 혹은 이미지 파일을 직접 다운로드 받는 것 보다 매우 작은 크기임을 확인할 수 있었다. 또한 시스템을 적용해야 일반적인 브라우저 환경과 비교하였을 때에 캐싱 시스템을 통해 네트워크가 제한된 환경에서 약 80%이상 빠른 속도로 웹 콘텐츠를 사용자에게 제공하는 것을 확인할 수 있었다.

웹 서비스를 사용자에게 제공하는 데에 있어 속도는 그 무엇보다도 중요한 요소 중 하나이며, 본 논문에서 제안한 시스템의 지속적인 연구를 통해 웹 서비스의 속도뿐만 아니라 네트워크 리소스 소비 양쪽 측면을 모두 고려한 서비스를 만드는 데에 일조할 수 있을 것으로 기대된다.

Acknowledgments

이 논문은 2018년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. 2018R1D1A3B07049722) 및 2019년도 한국기술교육대학교 교수 교육연구진흥과제 지원에 의하여 연구되었음

References

[1] P. Lepage, Your first progressive web app [Internet]. Available: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/?hl=en

[2] B. H. Andreas, T. A. Majchrzak, and T. M. Grønli. “Progressive web apps: The possible web-native unifier for mobile development,” in *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST)*, Porto: Portugal, pp, 344-351, April. 2017.

[3] Google developers, Nikkei achieves a new level of quality and performance with their multi-page PWA [Internet]. Available: https://developers.google.com/web/showcase/2018/nikkei

[4] Mozila MDN, Service worker api [Internet]. Available: https://developer.mozilla.org/en-US/docs/Web/api/Service_Worker_api

[5] N. Pande, A. Somani, S. P. Samal, and V. Kakkirala. “Enhanced web application and browsing performance through service-worker infusion framework,” in *2018 IEEE International Conference on Web Services (ICWS)*, San Francisco: CA, pp, 195-202, July. 2018.

[6] A. Gambhir, and G. Raj. “Analysis of cache in service worker and performance scoring of progressive web application,” in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Paris: France, pp, 294-299, June. 2018.

[7] Mozila MDN, Cache [Internet]. Available: https://developer.mozilla.org/en-US/docs/Web/api/cache

[8] Kravchenko and Maxim. Evaluation of security of serviceworker and related apis, Bachelor, Linnaeus University, Sweden, 2018.

[9] I. Malavolta, G. Procaccianti, P. Noorland, and P. Vukmirovic. “Assessing the impact of service workers on the energy efficiency of

progressive web apps," in *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, New Jersey: NJ, pp, 35-45, May, 2017.

[10] T. A. Majchrzak, B. H. Andreas, and T. M. Grønli. "Progressive web apps: the definite approach to cross-platform development?." in *Hawaii International Conference on System Sciences (HICSS)*, January, 2018.

김 현 국 (Hyun-Gook Kim)
2014년 2월 ~ 2017년 8월: 한국기술교육대학교 컴퓨터공학과 (공학사)
2017년 8월 ~ 현재: 한국기술교육대학교 컴퓨터공학과 석사과정
※관심분야 : 사물인터넷, 웹 어셈블리, 웹 표준

박 진 태 (Jin-Tae Park)
2005년 2월 ~ 2013년 8월: 한국기술교육대학교 컴퓨터공학과 (공학사)
2013년 9월 ~ 2015년 8월: 한국기술교육대학교 컴퓨터공학과 (공학석사)
2015년 9월 ~ 현재 : 한국기술교육대학교 컴퓨터공학과 박사과정
※관심분야 : Web Assembly, Standardization of Web Technologies, Web Application Engineering

최 문 혁 (Moon-Hyuk Choi)
2014년 2월 ~ 현재: 한국기술교육대학교 컴퓨터공학과 학사과정
※관심분야 : 인공지능, IoT

문 일 영 (Il-Young Moon)
2000년 2월 : 한국항공대학교 항공통신정보공학과 졸업 (공학사)
2002년 2월: 한국항공대학교 대학원 항공통신정보공학부 졸업 (공학석사)
2005년 2월: 한국항공대학교 대학원 정보통신공학과 졸업 (공학박사)
2004년 ~ 2005년 : 한국정보문화진흥원 선임연구원
2005년 3월 ~ 현재 : 한국기술교육대학교 컴퓨터공학부 교수
※관심분야 : 무선 인터넷 응용, 무선 인터넷, 모바일 IP